Adding Security to Implantable Medical Devices: Can We Afford It?

Muhammad Ali Siddiqi
Erasmus Medical Center,
Rotterdam, The Netherlands
m.siddiqi@erasmusmc.nl

Angeliki-Agathi Tsintzira
University of Macedonia,
Thessaloniki, Greece
angeliki.agathi.tsintzira
@gmail.com

Georgios Digkas
University of Macedonia,
Thessaloniki, Greece
g.digkas@uom.edu.gr

Miltiadis Siavvas
Information Technologies Institute,
Thessaloniki, Greece
siavvasm@iti.gr

Christos Strydis
Erasmus Medical Center,
Rotterdam, The Netherlands
c.strydis@erasmusmc.nl

Abstract

Implantable Medical Devices (IMDs) belong to a class of highly life-critical, resource-constrained, deeply embedded systems out there. Their gradual conversion to wirelessly accessible devices in recent years has made them amenable to numerous successful ethical-hacking attempts. These attacks were made possible due to the absence of proper security provisions in IMDs. IMD manufacturers have only very recently started taking cybersecurity threats seriously, a move that will force development teams to overhaul IMD designs and grow sharper reflexes in an industry that has historically opted for small, careful steps. Thus, valid concerns arise regarding the technical feasibility but, chiefly, the economic viability of adding security to IMDs. In this work, we assess the economic repercussions of securing IMDs by employing the concept of technical debt (TD) on the evolving IMD software. Our quantitative analysis reveals that security-related costs are currently well in hand, however, security-code TD amasses faster and will eventually overtake medical-code TD. The economic viability of IMDs will, thus, be ensured only if security-development efforts are allocated significant resources within the next decade.

Keywords

Technical debt, implantable medical device, IMD, security, embedded software

1 Introduction

In 2016, MedSec Holdings and Muddy Waters Research disclosed their findings concerning cybersecurity vulnerabilities discovered in St. Jude Medical implantable cardiac devices [7]. A bitter litigation war soon ensued between the two sides, and in the aftermath, the stock price of the manufacturer plummeted by 10 percent [7]. Although security flaws of undisclosed Implantable Medical Devices (IMDs) have been reported in the past, this was the first time a security firm went public without disclosing the issues first to the IMD manufacturer giving them due notice. Their rationale for doing so was – they claimed – to wake up the manufacturer and force them into action [7]. In the following years, cybersecurity issues in medical devices were reported by the US government to have risen significantly [11, 12, 13]. However, modern IMDs still lack essential security provisions, a situation that – if left unchecked – threatens to make IMDs very hazardous devices in the years to come.

In a world becoming rapidly conscious of cybersecurity attacks and the need for data privacy, which has led to serious steps like the EU General Data Protection Regulation 2016/679 (GDPR), the slow reflexes of the IMD industry can be attributed to a number of reasons.

Firstly, this has been a niche industry historically having *no concerns* for or expertise on cybersecurity aspects. The earliest guidance from the FDA on securing wireless medical devices was issued as late as 2013 [20]. Thus, this is still a transition period for IMD manufacturers and is, to a point, reasonable.

Secondly, the *highly resource-constrained nature* of IMDs could be perceived as prohibitive for incorporating mainstream security provisions, as necessitated by the modern cybersecurity landscape, while maintaining a high device autonomy and a small form factor [3, 24]. Fortunately, modern advances in embedded computing permit the ample use of such provisions without significantly impacting device autonomy (e.g., see [3] and [42] for IoT and IMDs, respectively).

Thirdly, the *steep (re)certification cost* of mission-critical and deeply embedded devices, such as IMDs are, is a major impediment to rehashing IMD design to include security provisions [18]. Though this is a valid concern, delaying incorporating IMD security provisions is a very short-sighted strategy in view of the prospective loss of life and of ensu-

International Conference on Embedded Wireless Systems and Networks (EWSN) 2021
17–19 February, Delft, The Netherlands
ISBN: 978-0-9949886-5-2

ing market sales due to successfully mounted cybersecurity attacks in the future. The St. Jude Medical case serves as a cautionary tale of this fact.

The fourth reason for being reluctant in adding security provisions to IMDs is, perhaps, the most serious and insidious one: Historically, the medical-functionality codebase of IMDs has been slow to change, driven by a need for high reliability and by the sheer fact that little functional change is necessary in such deeply embedded devices. However, imbuing IMDs with adept security involves the introduction of a secondary, security codebase from scratch. We anticipate that this new codebase – besides the aforementioned feasibility challenges – more crucially will impose a *change of pace in IMD code updates* but also in code *maintainability*. We foresee more frequent updates to cope (a) with the virtually unsecured IMD designs, and (b) with the rapidly expanding attack surface of IMDs since they are now wirelessly accessible via end-user smartphones, tablets and – indirectly – the Internet [47]. IMD security, thus, requires a design-paradigm shift and also is suspect to introducing new, perpetual costs for IMD-code maintenance, which should be covered by the IMD industry.

This final, hypothesized reason is the main drive behind the current paper, which looks at IMD security from a fresh angle: *Is it economically sustainable to add security to IMDs?* Convincing the manufacturers (and other stakeholders) that IMD security is economically viable and sustainable, will liberate a lot of security solutions that are considered at the moment "out of scope". Along the lines, it will also allow to invert the psychological bias of denial ("we do not need security") and complexity ("we can only add very trivial security due to costs"). We should stress that this work is an academic study and is in no way commissioned by or endorsing any third party whatsoever.

In order to address the IMD-economics question in a tangible way, we adopt in this work the systematic concept of *technical debt (TD)* for capturing the software costs of modern IMDs. We prefer TD over other methods (e.g., QMOOD [5] or CK [10]) because it covers a large variety of issues, ranging from code-convention violations to architectural problems. On top of that, the monetized nature of TD is proven to be a more helpful way to communicate maintainability benefits to non-software-engineering stakeholders, compared to the traditional software metrics.

TD is a software-engineering concept that expresses the implied development cost that is incurred due to taking shortcuts during software development in order to reduce the time-to-market of a product. The implied cost manifests itself as the *additional effort* that is required to maintain the resulting low-quality code [14]. As with regular (financial) debt, TD must also be paid back by improving embedded-code quality, e.g., by means of refactoring code, fixing code smells etc. The longer such improvements are delayed (or, equivalently, TD is not paid), the harder the development and maintenance of subsequent software releases that implement new functionality or features. It is exactly the new security features and their interplay with the existing medical-functionality ones that we wish to address in this work.

Hardware costs, even if feasible to capture via TD, would be mostly irrelevant for IMDs. Due to the IMDs' deeply-embedded nature, hardware changes never occur within a given device's lifetime (since it is implanted) and occur rarely within a given product line. Modern IMDs are, thus, software-driven devices (see Strydis [49, Chapter 2]), meaning that hardware changes incur virtually no TD. Besides, such changes can be captured via their repercussions in the respective software codebase, which changes far more frequently by comparison.

In order to perform TD analysis of IMD application code, we design a detailed experiment: Since getting access to sufficient large codebases from various IMD manufacturers is out of the question for legal and practical reasons, we carefully construct our own synthetic historical record of IMD-codebase changes, captured as software releases over time, targeting both *medical* and *security* aspects of those IMDs. The record has been made open-access and is freely available[1] to download and put under public scrutiny. It ranges from 1997 until 2028, so as to capture future IMD changes and, thus, make future predictions to help IMD stakeholders. This record will permit us to analyze the impact successive software releases have on IMDs in terms of TD amassed. With this work, we make the following novel contributions:

- A systematic analysis of security-related software costs in IMDs, based on software-TD analysis.
- Predictions of the TD impact of IMD medical and security codebases on future IMD costs.
- Along the lines, a short technical-feasibility study of inserting mainstream security mechanisms in commercial IMDs.

The rest of the paper is organized as follows. Background on the technical-debt concept is provided in Section 2, followed by an overview of related works in Section 3. In Section 4, we present our experiment design and, in Section 5, we provide the details of the various IMD-software versions developed for this study. TD is calculated based on these versions and evaluated in detail in Section 6; future predictions on cost are made. We conclude the discussion in Section 7.

2 TD Background & Employed toolflow

Technical Debt (TD) is composed of two parts: *principal* and *interest*. The principal is the amount of money a company has to pay in order to develop a software system to its optimal quality. If optimal quality has not been reached, this effectively means saving effort, or in other words, money. This amount (i.e., the principal) increases the company's capital, and can be invested in other activities. However, this *internal loaning* comes at a cost: any future *maintenance activity* on the codebase, e.g., for accommodating a new feature, will require increased effort due to the less-than-optimal code quality and maintainability. These additional efforts, are equivalent to paying an interest on a loan. In contrast to the financial interest, which is calculated at regular time intervals based on a given interest rate, TD interest is amassed only when the software artifact is being maintained.

Figure 1 illustrates the above concepts. Every design has the potential to reach an *optimal* quality level compared to

[1] https://gitlab.com/neurocomputing-lab/sims

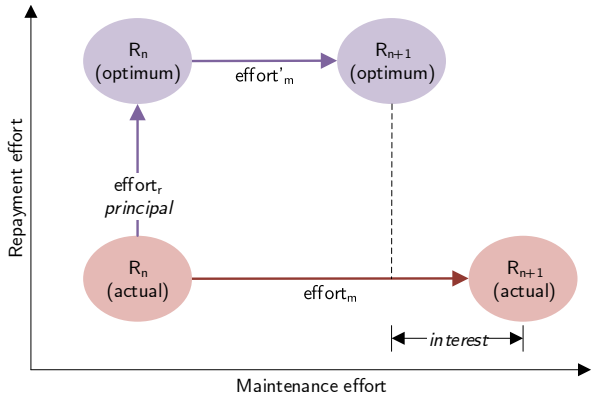


Figure 1. Relationship between TD principal and interest where R_n is the n^{th} design release [9]

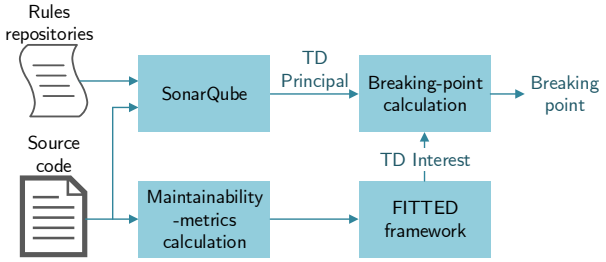


Figure 2. Tool flow employed for TD calculation

its *actual* level. In order to reach this level, the development team needs to dedicate some effort, which is equal to the **TD principal**. This activity, which involves code refactorings, is important for the repayment of TD, and hence, it is also called *repayment effort* ($effort_r$). On the other hand, the effort performed in order to add a new feature, enhance functionality or fix bugs is called *maintenance effort* ($effort_m$). In the case of maintaining an optimal design version, adding a feature requires $effort'_m$, whereas in the actual case, the same activity requires $effort_m$, which is always greater. The difference between these two efforts is the **TD interest**. It is important to note that $effort_r$ and $effort_m$ represent only *coding* and *verification* efforts.

2.1 Employed tools

In this work, the different code releases (i.e., intervals) of the IMD application are committed using the Git version-control system. TD principal is quantified via SonarQube, which uses the SQALE method [27] to measure the system TD (see Figure 2 and Algorithm 1). This tool first gets its input, i.e., file-level changes between these releases, through JGit, which is a Java library implementing Git. It then (1) reflects the application source code against a set of predefined rules, to identify violating code snippets, and (2) calculates the time required to resolve each violation. The total time required to fix all the violations represents the *TD principal*. This value, which is in *hours*, is converted into *currency* using a standard hourly rate (r_H) of **USD 45.81**, which is in line with the rate of an average developer in the US [54].

The *TD interest* can be calculated in various ways. In this work, it is calculated using FITTED, a framework for managing interest in technical debt [1, 9] (see Figure 2), which assesses TD interest by calculating the difference between the efforts required to *maintain* optimal and non-optimal software artifacts, respectively (see Figure 1). The metrics ($\mathbf{m} \in \mathbb{R}^4$) used for quantifying maintainability in order to calculate the above efforts are *coupling*, *cohesion*, *cyclomatic complexity* and *size* (in lines of code). They are calculated using the approach in [2] and are defined in Section 2.2. In FITTED, a *fitness* function $f : \mathbb{R}^4 \rightarrow \mathbb{R}$, is employed that takes the above metrics as input and returns the *fitness values* of the actual and optimal software artifacts, which are $f(\mathbf{m}_n)$ and $f(\mathbf{m}'_n)$, respectively, where $\mathbf{m}_n$ and $\mathbf{m}'_n$ are the non-optimal and optimal metrics belonging to the n^{th} code release. $effort_m$ and $effort'_m$ are directly proportional to the respective fitness values. TD interest (I) accumulated between releases $n-1$ and n can then be calculated using (1) [9]:

$$\begin{aligned} I_n &= effort_m - effort'_m \\ &= k_n \left(1 - \frac{f(\mathbf{m}'_n)}{f(\mathbf{m}_n)}\right), \forall\, n \in \mathbb{Z}^+ \end{aligned} \tag{1}$$

Here, k_n represents the lines of code that are added between the current (n) and the previous release ($n-1$).

Algorithm 1: TD Principal (P_n) calculation for a design release (codebase) R_n

Inputs : $n, R_n, rules, r_H$
Output: P_n
$N_R \leftarrow size(R_n)$;
$N_{rules} \leftarrow size(rules)$;
$hours \leftarrow 0$;
for $i \leftarrow 1$ **to** N_R **do**
  for $j \leftarrow 1$ **to** N_{rules} **do**
    $V_{i,j} \leftarrow$ total j^{th}-rule violations in the i^{th} file;
    $T_{i,j} \leftarrow$ time required to fix $V_{i,j}$;
  end
  $hours \leftarrow hours + T_{i,j}$;
end
return $P_n \leftarrow hours \times r_H$;

2.2 Metrics definitions

We now briefly describe the metrics referred to in this paper that are related to both TD principal and interest:

Accumulated TD: It is the sum of TD principal and interest, and thus, represents the total technical debt.

Coupling and Cohesion: Coupling indicates the number of dependencies between the files of a software project. The more the dependencies, the more difficult it becomes to maintain and extend a software system. The maintainability-metrics calculator determines coupling using (2), where FO_i (fan out) is the number of files referenced by the i^{th} file, and N_R is the total number of files in a code release.

$$coupling = \frac{1}{N_R} \sum_{i=1}^{N_R} FO_i \tag{2}$$

Cohesion represents the degree to which the lines inside a file interact with each other. It is calculated using the *lack of cohesion in methods (LCOM)* metric. If X_i is a set of line pairs in the i^{th} file that do not share any variable, and Y_i is a set of line pairs that have at least one common variable, then $LCOM_i$ can be calculated using (3) [10], where the maximum cohesion corresponds to the *LCOM* value of 0:

$$LCOM_i = \begin{cases} |X_i| - |Y_i|, & |X_i| > |Y_i| \\ 0, & otherwise \end{cases}, \forall\, i \in \{1, 2, \cdots, N_R\} \tag{3}$$

The overall cohesion of a code release is calculated using (4).

$$cohesion = \frac{1}{N_R} \sum_{i=1}^{N_R} LCOM_i \tag{4}$$

Low coupling and *high* cohesion are desirable qualities, indicating that the software is easy to understand, maintain and extend.

Cyclomatic complexity (CC): It refers to the number of independent paths throughout the code. Whenever the program control flow branches, e.g., due to an *if* statement, the cyclomatic complexity increases by one. The higher the CC, the harder the software becomes to understand, maintain and test: The larger the number of paths, the more the tests required to achieve a sufficient code test coverage.

Lines of code (LOC): This metric indicates the number of lines of software code that are not part of a comment. LOC can be used to estimate the programmers' productivity, during the development phase, or the software's maintainability during the production phase.

Breaking point: It is the point in the future (in terms of code releases) at which the cumulative TD interest[2] reaches and surpasses the TD-principal amount. At that point, all savings accumulated by not repaying TD will have been exhausted as a result of the additional maintenance effort during the software evolution [9]. The breaking point b_n for the n^{th} code release (R_n) is calculated using (5), where P_n is the TD principal at R_n:

$$b_n = \frac{P_n}{\sum_{i=1}^{n} I_i} \tag{5}$$

3 Related work

Technical debt is a widely used concept in software engineering. However, its use in improving software security has not been explored in detail [37]. Siavvas et al. [39] investigated the potential relationship between TD and software security, based on a relatively large repository of popular open-source software applications. Their preliminary findings suggest that TD, apart from quality issues, may potentially indicate the existence of security-vulnerability issues in a software. Similarly, TD has only recently been considered in energy-efficient software design for embedded systems. In this context, most of the emphasis has been on studying the impact of code refactoring on the energy consumption [33, 36]. Example domains include mobile applications [33] and vehicular technology [16]. However, the applicability of TD in IMD systems and other related domains, such as wireless body area networks (WBANs), has not been explored.

Fu [22] was the first to bring up TD in the context of medical devices. He pointed out that hackers can be regarded as the *messengers* of cybersecurity TD because they uncover the implications of the flaws that exist due to poor design choices. However, the discussion does not go in depth regarding the repercussions of amassed TD in view of securing future IMDs.

All in all, our work departs from the previous works by performing a comprehensive TD analysis to study the impact of adding security to modern IMD systems.

4 Experiment Design

We now explain our experiment design: In order to perform TD analysis of the IMD application code, we start by constructing a synthetic historical record of IMD design changes, captured as code releases over time, targeting both medical and security aspects of those IMDs. TD is not affected by exact years but, in order to also give readers a precise as possible timeline, this historical record dates back in the past as far as 1997 and extends to speculated future releases until 2028, so as to capture future IMD changes; see Table 1. This record will permit us to analyze the impact these software releases have on IMDs in terms of TD amassed.

An *ideal* TD analysis would require all the application-code releases to be coming from the IMD manufacturers. However, there are various obstacles to that approach:

1. There is no known repository that hosts application code from IMD manufacturers.
2. The sensitive nature of these products, coupled with the traditionally cryptic culture of the IMD industry, has made acquiring code sources directly from the manufacturers virtually impossible.
3. The other potential option is to reverse-engineer explanted IMDs. However – setting aside the ethical, legal and practical hurdles – this method will only give us access to the firmware binaries at best.

The above obstacles necessitate employing a *synthetic* codebase in the sense that the included IMD code has been synthetically created based on publicly available clinician's manuals (from multiple manufacturers), news articles, data sheets, and so on (see source(s) column in Table 1). This is a painstaking process and, yet, the only viable means of analyzing the IMD field currently undergoing a critical transition and drawing important conclusions for both the scientific and the industrial communities. Our confidence in the codebase representability is further safe-guarded by (a) employing auxiliary metrics (see Sections 6.1 and 6.2), and (b) making it publicly available in this work so as to encourage a critical review and improvement by the various IMD stakeholders.

[2]The cumulative TD interest is the sum of the current and all previous TD interests. It should not be confused with the accumulated TD.

Table 1. Overview of the constructed IMD timeline. The year, type of release, design-information sources and hardware modifications (if any) are shown.

Release	Year	Release type	Description of added design feature	Source(s)	Peripherals added*	
					Medical MCU	Security MCU
1	1997	Medical	Processor-based basic medical functionality	[55]	ADC, Cryotimer**	–
2	1999	Medical	TRX connection for configuration updates	[31, 35]	USART/SPI	–
3	1999	Medical	Read-out of sensor values	[29]	–	–
4	2001	Medical	Battery-level monitoring and read-out	[28]	–	–
5	2002	Medical	Safety modules, e.g., watchdog timer	[52]	Watchdog	–
6	2003	Medical	OTA-firmware-update support	[19, 46]	–	–
7	2008	Medical	Read-out of data logs with time-stamps	[53]	RTC	–
8	2017	Security	Fundamental security services	[30]	–	–
9	2020	Security	DoS-attack protection	[11, 13, 24, 50, 56, 17]	–	2 × USART/SPI
10	2023	Security	Replace SW cipher with a HW implementation	[43]	–	Crypto module
11	2026	Security	New security services	[34, 21, 41]	–	–
12	2027	Medical	Multi-sensory operation	[25]	–	–
13	2028	Security	Secure emergency mode	[38, 21, 17, 41]	–	Cryotimer

'–' : Not applicable or no change compared to previous.
* Core peripherals (e.g., clock- and energy-management units) not included.
** Ultra-low-energy timer of [43].

Next, we will go over the IMD classes considered, the selected hardware, and crucial assumptions made in setting up our experiment. A detailed presentation of the IMD code releases will be provided in Section 5, which is essential for motivating our results.

4.1 IMD applications

Two prominent IMD application classes are considered: neurostimulators and cardiac pacemakers. Cardiac implants hold the largest market share, whereas the neurostimulators are projected to witness the fastest growth. Roughly more than 50% of all IMDs in use belong to these two classes [23]. The hardware and software features included in this work largely capture the characteristics of actual IMDs. These features are inferred from publicly available information, from *multiple* IMD manufacturers, and are a good approximation for answering the research questions raised in this study.

In this work, the general closed-loop structure is kept the same in both the classes. One of the main differences is the sampling frequency (f_s) employed to capture the physiological signal. It has been shown that, for cardiac implants, f_s can be as low as 62.5 Hz [45] whereas, for neurostimulators, a f_s of 100 Hz is sufficient since most brain activity can be found within the 0–50 Hz range [51, 55]. The general structure of the application is based on the lightweight, wavelet-based filter design presented in [55]. Overall, we have encoded IMD software in C, which is consistent with the state of the art available throughout the assumed time period of study.

Although the doctor's *reader device* or the bedside base-station [48], are crucial components of the broader IMD system as well, in this study we strictly included IMD-application code, for two reasons: (i) IMDs are the bottleneck in terms of resources, e.g., their battery cannot be replaced during the operational lifetime. (ii) Most of the critical attacks, such as battery depletion, have a lasting effect on the IMD operation, and they are not targeted towards the reader.

4.2 IMD hardware platform

IMD manufacturers use commercial off-the-shelf (COTS) microcontrollers (MCUs) as their processing and/or controlling cores in modern IMDs [8]. To the best of our knowledge, these manufacturers do not design their own processors. In this work, the IMD-application source codes were tested on an EFM32 Tiny Gecko MCU, which is based on a 32-bit ARM Cortex-M0+ CPU [43] from Silicon Labs. In addition to being ultra-low power, the development kit and the integrated development environment (IDE) of this MCU come with Advanced Energy Monitoring, which enables live and accurate measurement of current draw. MCUs based on Cortex-M have been employed in latest commercial IMDs available, according to the official Bluetooth SIG listing of declared products and qualified designs [8]. Hence, this MCU is a suitable choice for our analysis. Moreover, for the costs associated with the wireless communication, a commercial implantable-grade transceiver, Microsemi ZL70103, has been used [31].

As will be shown in Section 6.1, the compiled code fits in the MCUs that were commercially available throughout the assumed time period of study. Moreover, the different application versions conform to the processing capabilities of such MCUs. The starting date for our analysis corresponds to the year when the 16-bit TI MSP430 – known to be used in IMDs – was first released [6, 38]. Although MCUs and microprocessors started appearing in commercial IMDs long before the MSP430 (e.g., RCA 1802 used in [26]), they did not come with C compilers. Since our TD analysis is only possible on applications written in C, MSP430 is an early enough and realistic starting point of our assumed timeline.

The above make it obvious that our hardware setup remains fixed throughout our experiments. This does not pollute our evaluation process since, as discussed in Section 1, hardware-caused TD is negligible. Conversely, pinning down the hardware platform used, allows for an even comparison of the different code releases. Finally, it should

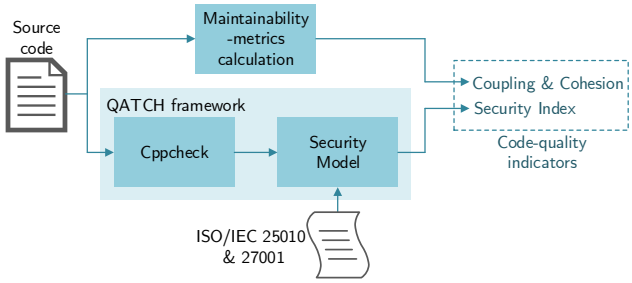


Figure 3. Tool flow employed for code-quality measurement

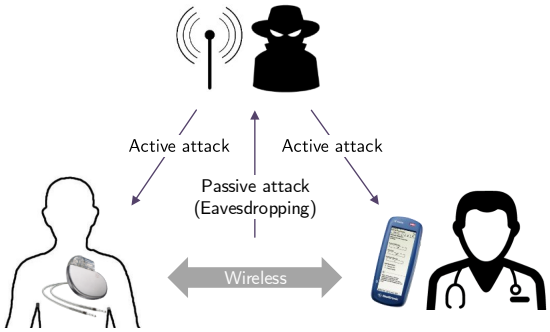


Figure 4. Threat model

be noted that the low-level, peripheral-support library provided by the MCU vendor and the cipher library (taken from a stable repository) are not included in the TD analysis since this code is not touched by the IMD developers under normal conditions.

4.3 Software-quality measurement

By fixing the hardware platform, we guarantee fairness at the hardware level. In order to keep the comparison of the different software releases also fair and minimally dependent on our coding skills, we employ *static code analysis* in order to determine whether the code-quality (and vulnerability) levels were maintained throughout the various releases.

We, thus, employ the QATCH framework [40], which is based on a static-code analyzer called Cppcheck (see Figure 3). The analyzer is configured to detect security issues that reside in the source code. A *security model* aggregates the results produced by Cppcheck based on a set of international quality and security standards (i.e., ISO/IEC 25010 and ISO/IEC 27001) and produces a single score, the security index, which reflects the internal security level of the analyzed software. Moreover, the coupling and cohesion metrics generated by the maintainability-metrics calculator (see Section 2.1) are also used for measuring maintainability, extendability and understandability of the different releases.

4.4 Threat model

To understand all hardware and software design choices made and captured in the codebase record, it is imperative to establish a threat model for the documented IMDs. Our threat model is very pessimistic and assumes an attacker with *full control* of the wireless channel between the reader and IMD, i.e., he/she can eavesdrop, modify, insert, block or replay messages between these two entities (see Figure 4). The attacker's aim could be to (i) modify or prevent patient treatment, (ii) steal patient data, or (iii) manipulate patient related data. As a result, the IMD-security system is required to provide certain security services, i.e., *confidentiality*, *integrity*, *authentication* and *availability*. In order to ensure user accountability, *non-repudiation* is also required. These will be further discussed in Section 5.

5 Timeline of IMD design releases

The *time period* of the IMD-codebase record extends from 1997 to 2028 (see Table 1). Without loss of generality, the timeline up to 2020 is mostly based on historical data, which is cited in detail. The timeline beyond 2020 is fictitious, yet is generated by the *conservative* inclusion of security features commonly proposed in current literature. Most post-2020 changes are security-related except for the use of multi-sensor recordings in next-generation neurostimulators to enable seizure prediction. This is used as an instance of a widely accepted medical design point in the future [25]. All in all, these design points are *not* the only choices available, but are taken as representatives of a general trend. It should be stressed that whether they will find their way in commercial IMDs or not does not affect the message of the paper, which is a cautionary tale: By blindly incorporating ever-expanding medical/security provisions in future IMDs, the economic repercussions for manufacturers will be dire.

In order to construct the historical timeline, we opted for *yearly* code commits as we found this to be a realistic time resolution. The timeline (of past code releases) is based on the earliest reported date in literature regardless of the implant class. This is also important for quantifying the TD impact of each individual feature (at each code commit), which would have been lost with coarser time resolution. We should stress, however, that the timeline resolution does not impact in any way the TD analysis; only the right sequence of code commits is relevant. Still, we chose to include particular timestamps so as to correlate with the historical IMD development (see Table 1).

In what follows, we detail the IMD code releases. This timeline is very comprehensive, yet is necessary for clearly documenting our steps and for providing a strong experiment basis. The interested reader can skip the remainder of this section, proceed to the results discussed in Section 6, and return here for more details on the code releases. We denote the releases by $R_{<release\,\#>}$, summarized in Table 1.

R_1: This code release implements the basic closed-loop medical functionality, as discussed in Section 4.1. A low-energy timer interrupt is used to wake up the MCU every $1/f_s$ seconds. The internal ADC is then used to sample the physiological signal. Upon processing the data to determine if the stimulus is needed or not, the MCU goes back to sleep.

R_2: In this release, an RF-communication interface is added to the IMD for configuration updates. For the example applications, the filter-coefficient values and the threshold values of the detection algorithm can be read and/or configured via the wireless interface. Moreover, the treatment can also be turned on or off. We took an implantable-grade transceiver (TRX) [31] as an example. This transceiver com-

municates with the MCU via an SPI interface, in which the MCU acts as the master. Upon receiving the data, the TRX sends a GPIO interrupt to the MCU so that it can retrieve it from the TRX buffer. Hence, additional code is added to R_1 in order to enable this wireless interface. It also includes the decoding of user commands, based upon which the IMD performs the required actions. Moreover, the IMD application also formats the data to be sent in bytes in order to use one of the MCU USARTs as the SPI master and does the opposite for the received data.

The release date corresponds to the year when the Medical Implant Communication Service (MICS) was created by the FCC and a separate band was allotted for IMD communication [35]. Although RF-communication capability in IMDs existed long before MICS, this year marks the first year of standardized implant communication.

R_3: The IMD is now able to emit basic data logs, such as the recorded ECG/ECoG values. In addition to determining the treatment status, these logs can also be used for device diagnostics and troubleshooting purposes. Among the earliest IMDs to do this were the Medtronic Kappa 400 series pacemakers [29].

R_4: The application can now get the voltage level of the battery via its ADC and send it to the reader when asked by the user. Moreover, it also includes an audio-tone-based notification system, to mimic the ones that exist in the vintage Medtronic GEM III series pacemakers [28], which alerts the patient when the battery level is too low. In this system, the application periodically measures the voltage level (daily in our examples) and determines if it is below a certain threshold. In case of a low battery level, a small speaker is enabled for 10 seconds via one of the MCU GPIO pins.

R_5: This release introduces a watchdog timer as a safety mechanism. The timer resets the system to recover from a faulty condition, which could be due to a design bug or an external event that puts the MCU in an unknown state, making it unresponsive. As an example, an MCU that is stuck during electrical stimulation can cause serious complications on the patient's health. Such timers can be found in many MCUs including the MSP430 series [52].

R_6: In case of a software bug or a major functionality change, the IMD firmware has to be updated. Manual firmware updates imply surgically explanting the IMD, which is a risky and costly endeavor. Therefore, ideally the implant should be able to update its firmware wirelessly; i.e., *over the air (OTA)*. Based on our review of the past FDA advisories, we found that the earliest prescribed IMD-firmware update was reported in 2003 for a St. Jude Medical pacemaker (ADx pulse generator) launched in the same year [19, 46].

In release R_6, the firmware update is made possible using an *application bootloader*. In contrast to a *standalone bootloader*, which directly overwrites the existing application image in the instruction memory through a serial interface such as UART or SPI, the application-bootloader update is a two-stage process. The existing application first downloads the new image (via the transceiver) into an external flash or a vacant portion in the main (internal) flash[3]. It then calls the application bootloader to validate the new firmware image and copy it from the download space to the code space in the internal flash. The advantage of using an application bootloader, especially in a life-critical medical device, is that any errors introduced during the downloading stage do not negatively impact the running application. This is because the entire image is downloaded and its integrity verified before starting the actual update [44].

R_7: This release implements more detailed data logs, which can be retrieved by the physician. These logs include the exact time stamps of certain events, e.g., epileptic seizures. This is made possible by using a real-time clock (RTC) module, which started appearing in some MSP430 parts (MSP430FG47x) around this time frame [53]. In this release, the user is also able to set the date and time of the device via the wireless interface.

R_8: As discussed in Section 1, due to the multiple reported vulnerabilities in IMD systems over the last decade or so and the strict measures taken by the FDA, we have finally started seeing standardized data-encryption implementations in these systems. For instance, the Azure pacemaker from Medtronic [30] implements NIST-standard encryption.

Release R_8 implements an ISO/IEC 9798-2-based, three-pass, mutual-authentication protocol, which is based on a pre-shared symmetric key between the reader and the implant. For data *confidentiality*, i.e., encrypting the reader commands and the IMD responses, the lightweight block-cipher SPECK is employed with block and key sizes of 64 and 128 bits, respectively. SPECK has been standardized in ISO/IEC 29167-22 as part of the RFID air interface standard (ISO/IEC 18000). For *authentication* and data *integrity*, a Cipher-based, Message-Authentication Code (CMAC) is employed, which generates a 32-bit MAC. Similarly, SPECK is used in *counter* mode to generate a fresh, 32-bit pseudo-random number (nonce) for replay protection. The interested reader can refer to [50] for a detailed description of the protocol and algorithms used.

It is important to note that we did not include the C-code implementation of SPECK in the TD analysis. This is because usually such code is taken from a stable repository and left untouched by the IMD developers.

R_9: Based on past ethical-hacking efforts on IMD systems, denial-of-service (DoS) attacks have entered the fray as one of the easiest attacks to mount [24, 11, 13]. In these attacks, which target the IMD *availability*, an attacker tries to send continuous connection requests in order to keep the IMD from performing its main functionality (Function-DoS), or to deplete the IMD battery in order to shut it down (Battery-DoS) [50].

One of the effective ways of protecting against Function-DoS is – next to the main, medical MCU – to introduce a second MCU in the IMD for handling communication packets and security. Battery-DoS can be prevented by initially operating this security MCU and the radio transceiver on the energy harvested from the incoming RF signal and allowing

[3]The choice of flash for downloading the image does not have an impact on TD since the corresponding change in the source code is negligible.

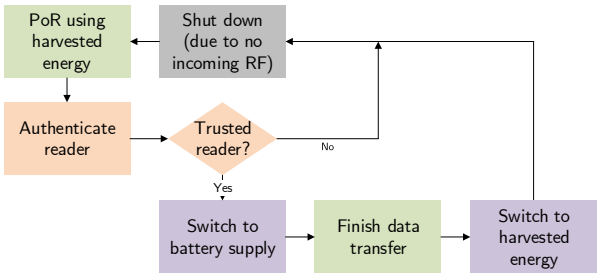


Figure 5. State machine of secondary, security MCU (PoR: Power-on reset)

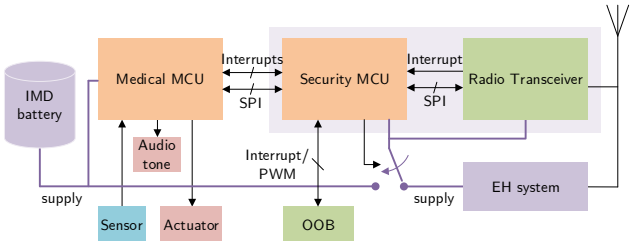


Figure 6. System overview of final IMD design, including DoS resistance (R_9) and emergency access (R_{13})

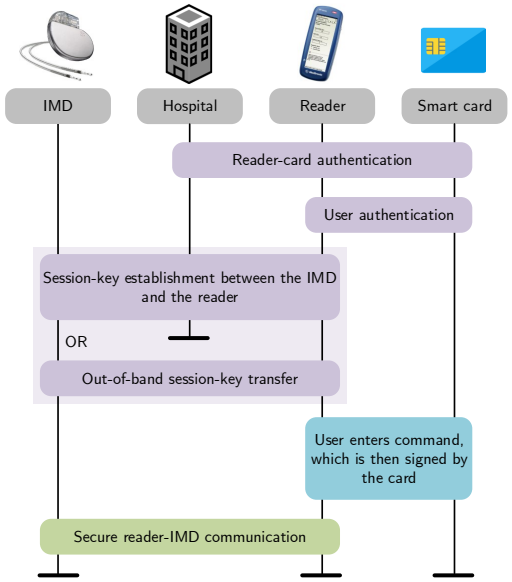


Figure 7. Overview of the security protocol employed in R_{11}-R_{13}

them to use the battery supply only after the external entity is authenticated. The rationale behind this approach is simple and can be conveyed from the finite state machine (FSM) of the security MCU, as shown in Figure 5. The updated IMD design is, then, shown in Figure 6. The signal to switch the security MCU and TRX power supply comes from the security-MCU GPIO pin, as shown in the figure. The two MCUs are connected via the SPI interface in which the security MCU acts as the master.

Various research works have advocated this dual-MCU approach in latest literature, [24, 50, 56, 17], which provides strong motivation for the industry to adopt in the future. Hence, in release R_9, a separate MCU is added to the IMD to act as the aforementioned security MCU (see Figure 6). As a result, this release onwards, we consider two separate C applications for TD analysis. It should be noted that the use of the watchdog timer from R_5 in the medical MCU also ensures that the medical treatment will be resumed in case communication is disrupted due to a disturbance in the wireless power transfer, which is a potential risk that R_9 introduces.

R_{10}: It is very much possible that a security primitive employed in an IMD becomes outdated after a certain time due to newly reported attacks on the primitive or due to the availability of better alternatives in terms of security, energy consumption and/or performance. To reflect this in our analysis, in release R_{10}, SPECK is replaced by the more secure AES-128. Many modern MCUs have a dedicated crypto peripheral that implements AES-128, among other primitives [43]. In this release as well, the security MCU uses its internal AES-128. It is important to note that even though a hardware implementation of the cipher is used, the required change will still be in software since the crypto peripheral sits within the MCU.

R_{11}: In the past, IMDs could only be accessed by the patient's physician. Modern IMDs, on the other hand, allow access to multiple users [30]. As a result, there is an increased possibility of medical mistakes, malpractice or even insider attacks. Therefore, *non-repudiation* is required to enforce user accountability. It ensures that a person is not able to deny their involvement in an IMD access if it negatively impacts the patient's health. Moreover, the current landscape also requires *access control* so that a user is only able to send commands to the IMD according to her allowed privileges. Finally, the possibility of using multiple readers implies that the security based on pre-shared keys is not practical. Hence, *secure-key management* is also needed.

Similarly to the DoS discussion for R_9, we observe an increased focus in recent literature on providing the above services due to the nature of the emerging threats. Many of these works, such as [34, 21, 41], propose the use of additional entities, i.e., a user smart card and a trusted third party. In R_{11}, one of these protocols [41] is implemented in order to provide the above security services. A brief overview of the protocol is shown in Figure 7. Non-repudiation is enabled by employing the signature of the command, which is signed by a personal smart card, and is sent to the IMD along with the command itself. Moreover, a hospital server is added to the overall system as a trusted third party in order to enable key management, access control and user authentication. Hence, this version requires the reader to be connected to Internet.

The IMD stores the signature so that it can be retrieved for dispute resolution in case the corresponding command results in the deterioration of patient's health. The signature must, therefore, be stored in a non-volatile memory, e.g., in the security-MCU flash memory, to protect against MCU resets.

R_{12}: Accurate prediction of epileptic seizures is an open topic in the neuroscience research. Neurostimulators are ideal candidates to enable such a treatment since they are

Table 2. Summary of IMD resource usage (R_{13} (2028))

	Lifetime* (years)		Delay** (ms)	Prog.-memory
	Neuro	Cardiac		footprint (kB)
Without Security	8.7	14.6	20.3	27.5
With Security	7.1	11.7	85.9	53.1§

* For a typical IMD battery size of 9.5 Ah.
** It includes security-processing and TRX (SPI data handling and RF transmission) delays pertaining to a communication session in which 256 bytes of filter coefficients are read from the IMD.
§ It includes the program-memory footprint of *both* MCUs.

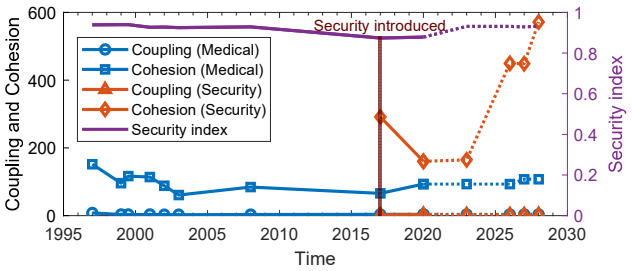


Figure 8. Overview of the code quality across all the IMD releases for both the medical and the security codebases.

already used for seizure suppression. One of the prominent research directions is to add multiple sensors to the implant in order to improve the prediction accuracy [25]. In order to capture any demanding future medical enhancement, R_{12} mimics the above scenario in which the closed-loop IMD system is based on multi-sensor inputs.

In this version, the MCU ADC is used in *scan* mode to sample multiple sources. In order to process these samples, a separate filtering operation per each added input source is required. Since only one MCU is employed for the medical application, these executions have to be performed sequentially, which increases the active-vs-sleep duty cycle of the MCU.

R_{13}: Another important security feature that is touted in modern literature is *emergency access* [38, 21, 17], which does not exist in IMDs at present. This feature allows paramedics, which are unknown to the patient, secure access to the IMD in an emergency scenario. Since the paramedic reader and IMD do not share a secret, the protocol from R_{11} can still be used. However, it cannot work in the absence of an Internet connection, which can help establish trust remotely.

Release R_{13} solves this problem by using an out-of-band (OOB) channel, such as galvanic coupling, ultrasound communication etc. This channel is used to pair the reader and the IMD by transferring a fresh symmetric key to enable secure communication (see Figure 7). One approach is to employ ASK modulation along with PWM encoding of bits in the reader-IMD OOB channel. Bits 0 and 1 can be differentiated by choosing different PWM duty cycles for each. The security MCU wakes up (via an interrupt) on the rising edge of every received bit. It then records the value on the same GPIO pin after a certain time period (with the help of a timer) in order to determine the bit level (1 or 0). The OOB data rate does not have to be high since the volume of data to be transferred, i.e., the session key, identifiers and nonces, is very low. The system architecture of the final IMD design is shown in Figure 6.

6 Experimental Results

In this section, we present the results of the TD analysis, which tries to capture the repercussions of introducing security at a certain point in the IMD-development timeline, and its interplay with the medical application.

6.1 Checking technical feasibility

We first briefly perform an IMD-autonomy and -performance analysis of the final design (R_{13} (2028)) in order to ensure that our experimental code does not introduce prohibitive energy, processing and area overheads. For this analysis, the MCU-processor clock frequency is set to the default value of 19 MHz, whereas the effective data rate of the transceiver is set to the maximum of 265 kbps. The supply voltage of the setup is set to 3.3 V. It is assumed that the transceiver is involved in 3 minutes of active data communication per day, which corresponds to the worst-case behavior compared to a commercial bedside reader [48]. Moreover, the pacemaker energy consumption during stimulation is assumed to be 20 μJ per heartbeat, based on worst-case figures from commercial devices [15]. The neurostimulator energy consumption during stimulation is borrowed from an actual seizure-suppression system [32]: under worst-case conditions, the stimulation current, pulse width, pulse frequency and burst duration are assumed to be 12 mA, 1 ms, 333 Hz and 10 seconds, respectively, with an average of 4.3 seizures per day [4].

The results are summarized in Table 2, which shows that the IMD resource usage is within acceptable limits: the calculated battery lifetime is sufficiently long, the security-processing and communication delays are negligible, and the program-memory footprints are small. These results also agree with previous findings [42, 41] that security does not significantly impact IMD autonomy.

6.2 Checking code quality

Figure 8 provides the results of the static code analysis introduced in Section 4.3. The `security index` is high across all the releases, which means that the changes that were made throughout these versions do not introduce new code vulnerabilities. Moreover, `coupling` and `cohesion` values stay fairly constant across all the releases, indicating that a consistent code quality was kept throughout the analyzed timeline.

6.3 Technical-debt analysis

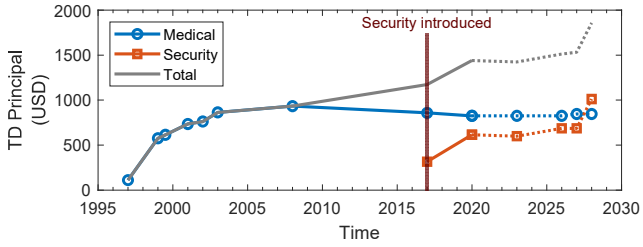
The TD principal, interest and accumulated TD of the IMD application-code releases are summarized in Figure 9. As mentioned in Section 2.1, an hourly rate of USD 45.81 is used for the TD calculations. It should be clarified that the TD costs correspond to *additional* repayment and maintenance activities and they do not represent the total development costs, as previously illustrated in Figure 1; these activities correspond to coding and verification efforts only.

We notice a steep increase in the *TD principal* at R_2 (1999), since therein is implemented the serial interface

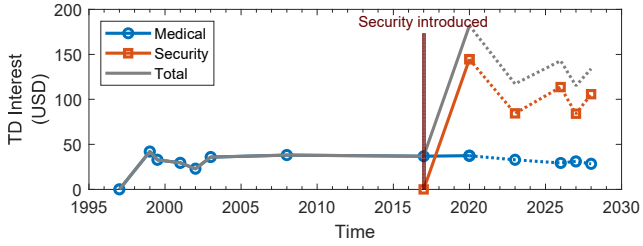
with the transceiver and the command decoding. This, interestingly, indicates that the wireless-interface-related code forms the major component of the application instead of the medical functionality. Moreover, because of this large increase, the next release (R_3 (1999)) causes a relatively steeper rise in the *TD interest* due to the increased maintenance effort. The decline in the TD principal of the medical application at R_8 (2017) and R_9 (2020) is because the communication-related processing in the medical code was moved to the security code. However, this reduction does not match the corresponding rise in the security-code principal during the same period due to the security-protocol implementation. What is more, two serial interfaces – one to the transceiver and one to the medical-application MCU – are added in R_9 (2020). It is important to note that the observed rise in the security-code TD principal does not include the cipher library in the analysis, as mentioned in the R_8 description. As a result, we do not see any noticeable change in the principal costs when replacing the cipher (i.e., SPECK with AES-128) since only the associated wrapper functions required change (R_{10} (2023)).

Having explained the reasons behind the morphology of the TD curves, let us now take a step back and assess the information they offer us. We can see that the total `TD principal` (grey line) reaches a maximum cost of just under USD 1,900, whereas the total `TD interest` for individual releases stays below USD 190. From either of the two curves, it can be deduced that *the security code is indeed more costly to extend and maintain than the medical code*. The interest, especially, is at least double for the security component. As a result, the `accumulated TD` (Figure 9c) is mostly driven by that component. By inspecting the total trend line, it is also interesting to notice that *security-driven code changes will eventually overtake medical-driven ones in the future*. Yet, we should pay attention to the actual cost these changes incur, as predicted by the analysis tools: Accumulated TD reveals that additional IMD-code repayment and maintenance costs are limited to only a few thousand dollars in the near future but those can *drastically deteriorate* for IMD manufacturers in the longer term, if left unchecked.

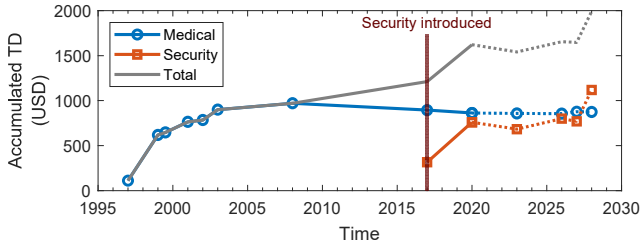
In Figures 10a and 10b, two of the components for calculating the TD interest have also been plotted: cyclomatic complexity and lines of code, so as to offer more insights on our application behavior. We see that the `cyclomatic complexity (CC)` of the security code increases at a faster rate than that of the medical code due to the type of complexity involved in the respective applications: Even a minimal change in the security protocol results in a significant increase in the corresponding FSM complexity (see Figure 5). For instance, numerous fallbacks are added so that the IMD FSM returns to a stable state in case the communication is disrupted midway. Besides, a significant portion of the IMD code is composed of control-flow statements. As a result, the `Lines-of-Code (LOC)` curve is very similar to the cyclomatic-complexity curve. Note that the LOC values seem to be relatively low. This is because the low-level peripheral-support library from the MCU vendor and the cipher library are not included in the TD analysis since they are not modified by the IMD developers.



(a) TD Principal

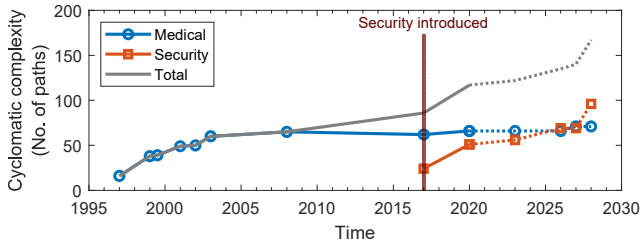


(b) TD Interest

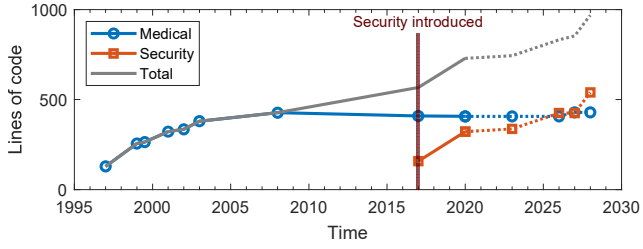


(c) Accumulated TD

Figure 9. Overview of total and per-IMD codebase (medical, security) TD metrics (see Section 2.2 for metrics definitions). Solid lines indicate existing, documented IMD-code features, while dotted lines show future, projected features. Costs are calculated based on default SonarQube hourly rate ($45.81).



(a) Cyclomatic complexity



(b) Lines of code

Figure 10. Overview of CC and LOC for the security and medical codebases.

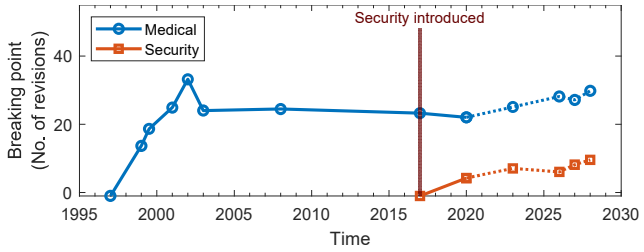


Figure 11. Overview of the breaking point for the security and medical codebases.

The higher CC of the security code is, finally, also reflected in the `breaking-point` curves plotted in Figure 11; The potential breaking point of the security code, at a given point in time, can be reached significantly earlier than that of the medical code. This observation indicates that the security code should be developed after careful planning. It also tells us that the medical code is easier to maintain since its breaking-point curve is always higher than the security curve.

6.4 Discussion

The work in this paper was carried out in order to answer the critical question whether adding security to modern IMDs is an economically viable and sustainable venture for IMD manufacturers (and other stakeholders). In the face of a rising number of cybersecurity attacks, this question becomes very relevant and time-sensitive.

The analysis has revealed that *adding* security code to an IMD medical-only codebase is going to be more difficult (in effort, and thus in cost) than adding new medical code, as the TD-principal estimations reveal (Figure 9). It will also be more difficult to *maintain* the security code compared to the medical one. These difficulties translate to higher development costs, which stem from the fact that the security codebase is generally more complex, more volatile and can deteriorate or break more easily. Fortunately, such software costs are rather low and can be shouldered by manufacturers.

Our analysis has necessarily relied on a synthetically constructed codebase; however, should our TD projections be accurate, the *security-driven TD can become critical in the future*. This finding is worrisome given that the security provisions of future IMD systems will grow to encompass also IMD readers (see Figure 7) and even remote IMD-company servers, each extra component introducing its own security codebase. In this context, security-driven TD is expected to rise even more steeply. Therefore, unlike the medical codebase (which in many cases remains practically unchanged across IMD generations), the security codebase has to be frequently refactored for the overall TD to remain in check.

The above findings lead to the main conclusion that present-day IMDs can be financially tractable with (perhaps necessarily) "quick and dirty" security solutions but this modus operandi has to transition soon to a more structured security-development approach so as to keep development costs under control and, thus, the viability of future IMD systems high for IMD manufacturers, insurance companies, healthcare systems and, eventually, patients themselves.

7 Conclusions

In the recent past, there has been a significant ramp-up in IMD ethical-hacking activities. The regulatory bodies worldwide are also increasing their pressure on the IMD manufacturers to improve the security of these devices. In this work, we embarked on a methodology to quantitatively analyze the cost of adding security in the existing devices from the perspective of embedded-software technical debt (TD). This is the first time that TD, which is a relatively new concept, has been used to analyze this class of embedded systems. By necessarily relying on a synthetically constructed IMD codebase, we found that security software, on one hand, is costlier to develop and maintain than the preexistent, purely medical software in IMDs but overall costs are insignificant in the short term. On the other hand, the higher complexity and volatility of the security codebase is projected not only to dominate future costs but also to disrupt the economic viability of IMD products in the next decade, if the IMD-software TD growth is left unchecked.

8 Acknowledgments

This work has been supported by the EU-funded project SDK4ED (Grant Agreement No. 780572) and would not have been completed without the invaluable feedback of Prof. Christian Doerr, Dr. Apostolos Ampatzoglou, Prof. Alexandros Chatzigeorgiou and Prof.dr.ir. Paris Avgeriou.

9 References

[1] A. Ampatzoglou, A. Ampatzoglou, P. Avgeriou, and A. Chatzigeorgiou. Establishing a framework for managing interest in technical debt. In *5th International Symposium on Business Modeling and Software Design, BMSD*, 2015.

[2] E.-M. Arvanitou, A. Ampatzoglou, A. Chatzigeorgiou, and P. Avgeriou. Software metrics fluctuation: a property for assisting the metric selection process. *Information and Software Technology*, 72:110–124, 2016.

[3] J.-P. Aumasson and A. Vennard. Cryptography in industrial embedded systems: our experience of needs and constraints. In *NIST Lightweight Cryptography Workshop 2019*. NIST, 2019.

[4] M. Balish, P. S. Albert, and W. H. Theodore. Seizure frequency in intractable partial epilepsy: a statistical analysis. *Epilepsia*, 32(5):642–649, 1991.

[5] J. Bansiya and C. G. Davis. A hierarchical model for object-oriented design quality assessment. *IEEE Transactions on software engineering*, 28(1):4–17, 2002.

[6] S. Barrett and D. Pack. *Microcontroller Programming and Interfacing TI MSP430*. Number pt. 1 in Synthesis Lectures on Digital Circuits and Systems. Morgan & Claypool Publishers, 2011.

[7] V. Blue. Turns out, pacemaker security is terrifying [online]. 2017. URL: `https://www.engadget.com/2017-04-21-pacemaker-security-is-terrifying.html`.

[8] Bluetooth SIG. View previously qualified designs and declared products [online]. 2020. URL: `https://launchstudio.bluetooth.com/Listings/Search`.

[9] A. Chatzigeorgiou, A. Ampatzoglou, A. Ampatzoglou, and T. Amanatidis. Estimating the breaking point for technical debt. In *2015 IEEE 7th International Workshop on Managing Technical Debt (MTD)*, pages 53–56. IEEE, 2015.

[10] S. R. Chidamber and C. F. Kemerer. A metrics suite for object oriented design. *IEEE Transactions on software engineering*, 20(6):476–493, 1994.

[11] CISA. ICS Advisory (ICSMA-17-241-01) [online]. 2017. URL: `https://www.us-cert.gov/ics/advisories/ICSMA-17-241-01`.

[12] CISA. ICS Advisory (ICSMA-18-179-01) [online]. 2018. URL: https://www.us-cert.gov/ics/advisories/ICSMA-18-179-01.

[13] CISA. ICS Medical Advisory (ICSMA-19-080-01) [online]. 2020. URL: https://www.us-cert.gov/ics/advisories/ICSMA-19-080-01.

[14] W. Cunningham. The WyCash portfolio management system. *ACM SIGPLAN OOPS Messenger*, 4(2):29–30, 1992.

[15] M. Deterre. *Toward an energy harvester for leadless pacemakers*. Theses, Université Paris Sud - Paris XI, July 2013. URL: https://tel.archives-ouvertes.fr/tel-00868838.

[16] U. Eliasson, A. Martini, R. Kaufmann, and S. Odeh. Identifying and visualizing architectural debt and its efficiency interest in the automotive domain: A case study. In *2015 IEEE 7th International Workshop on Managing Technical Debt (MTD)*, pages 33–40. IEEE, 2015.

[17] N. Ellouze, S. Rekhis, N. Boudriga, and M. Allouche. Powerless security for cardiac implantable medical devices: Use of wireless identification and sensing platform. *Journal of Network and Computer Applications*, 107:1–21, 2018.

[18] Emergo. Compare the time, cost and complexity of getting regulatory approval for medical devices [online]. 2017. URL: https://www.emergobyul.com/resources/worldwide/global-regulatory-comparison-tool.

[19] FDA. ADX Pulse Generator Firmware Anomaly Correction. *Devices@FDA*, 2003.

[20] FDA. Radio Frequency Wireless Technology in Medical Devices - Guidance for Industry and FDA Staff. *Guidance Document*, 2013.

[21] C. Fu, X. Du, L. Wu, Q. Zeng, A. Mohamed, and M. Guizani. POKs Based Secure and Energy-Efficient Access Control for Implantable Medical Devices. In *Security and Privacy in Communication Networks*, pages 105–125, 2019.

[22] K. Fu. On the Technical Debt of Medical Device Security. In *Frontiers of Engineering: Reports on Leading-Edge Engineering from the 2015 Symposium*. National Academies Press, 2016.

[23] Grand View Research. *Microelectronic Medical Implants Market Analysis Report by Product, by Technology, And Segment Forecasts, 2018 - 2025*, 2018. URL: https://www.grandviewresearch.com/industry-analysis/microelectronic-medical-implants-market.

[24] 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 *2008 IEEE Symposium on Security and Privacy (sp 2008)*, pages 129–142. IEEE, 2008.

[25] L. Kuhlmann, K. Lehnertz, M. P. Richardson, B. Schelter, and H. P. Zaveri. Seizure prediction—ready for a new era. *Nature Reviews Neurology*, 14(10):618–630, 2018.

[26] M. E. Leckrone and V. T. Cutolo Jr. Multi-mode microprocessor-based programmable cardiac pacer, Dec. 4 1984. US Patent 4,485,818.

[27] J.-L. Letouzey. The SQALE method for evaluating technical debt. In *2012 Third International Workshop on Managing Technical Debt (MTD)*, pages 31–36. IEEE, 2012.

[28] Medtronic. *GEM® III VR 7231 Implantable Cardioverter Defibrillator - System Reference Guide*, 2001.

[29] Medtronic. *Kappa® 400 series and DX2 pacemakers Model 9952 - Volume II, Pacemaker Reference Guide*, 2001.

[30] Medtronic. *Azure™ S SR MRI SureScan™ W3SR01 - Device Manual*, 2017.

[31] Microsemi. *ZL70103 Medical Implantable RF Transceiver - Datasheet, Revision 2*, 2015.

[32] NeuroPace. *RNS® System User Manual*, 2019.

[33] F. Palomba, D. Di Nucci, A. Panichella, A. Zaidman, and A. De Lucia. On the impact of code smells on the energy consumption of mobile applications. *Information and Software Technology*, 105:43–55, 2019.

[34] C.-S. Park. Security mechanism based on hospital authentication server for secure application of implantable medical devices. *BioMed research international*, 2014, 2014.

[35] Pike and I. Fischer. *Communications Regulation*. Communications Regulation. Pike & Fischer, 2003.

[36] G. Pinto, F. Soares-Neto, and F. Castor. Refactoring for energy efficiency: A reflection on the state of the art. In *2015 IEEE/ACM 4th International Workshop on Green and Sustainable Software*, pages 29–35. IEEE, 2015.

[37] K. Rindell, K. Bernsmed, and M. G. Jaatun. Managing security in software: Or: How i learned to stop worrying and manage the security technical debt. In *Proceedings of the 14th International Conference on Availability, Reliability and Security*, pages 1–8, 2019.

[38] 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 *2014 IEEE Symposium on Security and Privacy*, pages 524–539. IEEE, 2014.

[39] M. Siavvas, D. Tsoukalas, M. Janković, D. Kehagias, A. Chatzigeorgiou, D. Tzovaras, N. Aničić, and E. Gelenbe. An empirical evaluation of the relationship between technical debt and software security. In *9th International Conference on Information Society and Technology*, 2019.

[40] M. G. Siavvas, K. C. Chatzidimitriou, and A. L. Symeonidis. Qatch-an adaptive framework for software product quality assessment. *Expert Systems with Applications*, 86:350–366, 2017.

[41] M. A. Siddiqi, C. Doerr, and C. Strydis. IMDfence: Architecting a Secure Protocol for Implantable Medical Devices. *IEEE Access*, 8:147948–147964, 2020.

[42] M. A. Siddiqi and C. Strydis. IMD security vs. energy: are we tilting at windmills? POSTER. In *Proceedings of the 16th ACM international conference on computing frontiers*, pages 283–285, 2019.

[43] Silicon Labs. *EFM32 Tiny Gecko 11 Family - Reference Manual*, 2018.

[44] Silicon Labs. *UG103.6: Bootloader Fundamentals*, 2020.

[45] F. Simon, J. P. Martinez, P. Laguna, B. van Grinsven, C. Rutten, and R. Houben. Impact of sampling rate reduction on automatic ecg delineation. In *2007 29th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, pages 2587–2590. IEEE, 2007.

[46] St. Jude Medical. *Product Performance Report - Cardiac Rhythm Management*, 2006.

[47] St. Jude Medical. *Clinician Programmer App For Spinal Cord Stimulation Systems Model 3874 - Clinician's Manual*, 2015.

[48] St. Jude Medical. *FAQs - Merlin.net™ Patient Care Network (PCN) 8.0 Q&A*, 2015.

[49] C. Strydis. *Universal Processor Architecture for Biomedical Implants: The SiMS Project*. PhD thesis, Delft University of Technology, Delft, Netherlands, March 2011.

[50] C. Strydis, R. M. Seepers, P. Peris-Lopez, D. Siskos, and I. Sourdis. A system architecture, processor, and communication protocol for secure implants. *ACM Transactions on Architecture and Code Optimization (TACO)*, 10(4):1–23, 2013.

[51] W. O. Tatum. Ellen R. Grass Lecture: Extraordinary EEG. *The Neurodiagnostic Journal*, 54(1):3–21, 2014.

[52] Texas Instruments. *MSP430F15x, MSP430F16x, MSP430F161x Mixed Signal Microcontroller - Datasheet*, 2002.

[53] Texas Instruments. *MSP430FG47x Mixed Signal Microcontroller - Datasheet*, 2008.

[54] U.S. Bureau of Labor Statistics. May 2019 National Occupational Employment and Wage Estimates United States [online]. 2019. URL: https://www.bls.gov/oes/current/oes_nat.htm.

[55] M. N. van Dongen, A. Karapatis, L. Kros, O. E. Rooda, R. M. Seepers, C. Strydis, C. I. De Zeeuw, F. E. Hoebeek, and W. A. Serdijn. An implementation of a wavelet-based seizure detection filter suitable for realtime closed-loop epileptic seizure suppression. In *2014 IEEE Biomedical Circuits and Systems Conference (BioCAS) Proceedings*, pages 504–507. IEEE, 2014.

[56] Q. Yang, S. Mai, Y. Zhao, Z. Wang, C. Zhang, and Z. Wang. An on-chip security guard based on zero-power authentication for implantable medical devices. In *Circuits and Systems (MWSCAS), 2014 IEEE 57th International Midwest Symposium on*, pages 531–534. IEEE, 2014.