UnB - UNIVERSIDADE DE BRASÍLIA

FGA - FACULDADE GAMA

PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA BIOMÉDICA

BOAS PRÁTICAS NO CICLO DE VIDA PARA MELHORIA DA SEGURANÇA DE *SOFTWARE* PARA DISPOSITIVOS MÉDICOS

DIOGO DE CARVALHO RISPOLI

ORIENTADORA: Dra. LOURDES MATTOS BRASIL

DISSERTAÇÃO DE MESTRADO EM ENGENHARIA BIOMÉDICA

PUBLICAÇÃO: 010A/2013

BRASÍLIA/DF: JUNHO – 2013

UnB - UNIVERSIDADE DE BRASÍLIA

FGA - FACULDADE GAMA

PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA BIOMÉDICA

BOAS PRÁTICAS NO CICLO DE VIDA PARA MELHORIA DA SEGURANÇA DE *SOFTWARE* PARA DISPOSITIVOS MÉDICOS

DIOGO DE CARVALHO RISPOLI

DISSERTAÇÃO DE MESTRADO SUBMETIDA AO PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA BIOMÉDICA DA FACULDADE GAMA DA UNIVERSIDADE DE BRASÍLIA, COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM ENGENHARIA BIOMÉDICA.

APROVADA POR:

_____

Profa. Dra. Lourdes Mattos Brasil
Orientadora

_____

Prof. Dr. Georges Daniel Amvame Nze
Examinador Interno

_____

Prof. Dr. Fabricio Ataides Braz
Examinador Externo

BRASÍLIA/DF, 06 DE JUNHO DE 2013.

FICHA CATALOGRÁFICA

RISPOLI, DIOGO DE CARVALHO
Boas práticas no ciclo de vida para melhoria da segurança de *software* para dispositivos médicos, [Distrito Federal] 2013.
127p., 210 x 297 mm (FGA/UnB Gama, Mestre, Engenharia Biomédica, 2013). Dissertação de Mestrado - Universidade de Brasília. Faculdade Gama. Programa de Pós-Graduação em Engenharia Biomédica.
1. Segurança da Informação 2. Segurança de *Software*
3. Ciclo de Vida do *Software* Médico 4. *Hackers*
I. FGA UnB Gama/ UnB. II. Título (série)

REFERÊNCIA BIBLIOGRÁFICA

RISPOLI, D. C. (2013). Boas práticas no ciclo de vida para melhoria da segurança de *software* para dispositivos médicos. Dissertação de Mestrado em Engenharia Biomédica, Publicação 010A/2013, Programa de Pós-Graduação em Engenharia Biomédica, Faculdade Gama, Universidade de Brasília, Brasília, DF, 127p.

CESSÃO DE DIREITOS

AUTOR: DIOGO DE CARVALHO RISPOLI.
TÍTULO: BOAS PRÁTICAS NO CICLO DE VIDA PARA MELHORIA DA SEGURANÇA DE *SOFTWARE* PARA DISPOSITIVOS MÉDICOS.

GRAU: Mestre

ANO: 2013

_____

2013
STN LOTE L BLOCO D APTO 119, COND. TOSCANA – BAIRRO ASA NORTE.
CEP 70.770-100 Brasília, DF – Brasil.

DEDICATÓRIA

Para Carolina, com todo amor.

Toda tarde digo para mim mesmo: afinal,
eis o meu mundo.

O mesmo beijo, o mesmo quarto claro, com seu assoalho brilhante refletindo o meu passo;
as mesmas paredes brancas me envolvendo com afáveis gestos de paz; o mesmo rádio silencioso, entre livros empilhados, a mesma estante fechada que a um gesto meu descobre tesouros como velha mala de pirata.

Afinal, eis o meu mundo.
A mesma insubstituível companhia,
a mesma presença até
quando longe dos olhos,
a mesma voz perguntando, a mesma voz respondendo,
o mesmo odor suave da janta, do tempero cozinhado,
a mesma impressão de quem chega de ombros nus e veste
ajudado um macio agasalho.

Afinal, eis o meu mundo.
Como o pescador solitário, diante das vagas:
- eis o meu mar.

Como o pássaro do dilúvio diante do primeiro ramo:
- afinal, eis a terra !

Meu Mundo – J. G. de Araujo Jorge.

AGRADECIMENTOS

Agradeço ao professor Vinicius Rispoli, meu irmão, por todo apoio que me deu ao longo do desenvolvimento deste trabalho. Agradeço também, pelo mesmo motivo, a minha esposa Carolina Girardi que acompanhou e sofreu tudo junto comigo. A dedicação de vocês foi fundamental para que eu conseguisse chegar até aqui.

Sou muito grato à minha orientadora, professora Lourdes Mattos Brasil, pela liberdade com que me deixou tratar o tema de mestrado que escolhi desde o pré-projeto.

Ao professor Marcelino Andrade, que me deu um grande incentivo para que eu me juntasse ao Programa de Pós-Graduação em Engenharia Biomédica da FGA/UnB, o meu muito obrigado.

Também agradeço ao Grupo de Engenharia de *Software* do Departamento de Ciência da Computação da Universidade de Brasília, aqui representados pela Paula Fernandes, por ceder o *software* para o estudo de caso desta pesquisa.

Finalmente, sou grato ao Adriano Carmo e aos meus colegas de trabalho, que me deram um grande apoio para que pudesse concluir este mestrado. Aos revisores e apoiadores anônimos, aqueles que de uma forma ou outra contribuíram para o resultado final deste trabalho, agradeço fortemente a ajuda. Não esquecerei o auxilio de vocês.

RESUMO

BOAS PRÁTICAS NO CICLO DE VIDA PARA MELHORIA DA SEGURANÇA DE *SOFTWARE* PARA DISPOSITIVOS MÉDICOS

Autor: DIOGO DE CARVALHO RISPOLI
Orientadora: Profa. Dra. Lourdes Mattos Brasil
Programa de Pós-Graduação em Engenharia Biomédica
Brasília, Junho de 2013.

Apesar do uso generalizado da tecnologia da informação na área médica, pouca atenção é dada à segurança do *software* que executa ou coleta informações de pacientes em dispositivos médicos. Falhas de segurança de *software* nestes dispositivos podem levar a problemas que vão desde a revelação não autorizada da informação médica do paciente até lesões ou mortes. Com o objetivo de melhorar a qualidade e aumentar a segurança deste tipo de equipamento, este trabalho propõe uma metodologia para inclusão de atividades de segurança no ciclo de vida do *software* para equipamentos médicos, especialmente na sua fase de construção e testes. Esta metodologia foi desenvolvida a partir da avaliação de normas para construção de *software* para dispositivos médicos e sua correlação com técnicas de construção de *software* seguro. A partir da adoção de técnicas de desenvolvimento *seguro*, atividades como casos de abuso, análise de riscos, revisão de código-fonte e testes de penetração passam a fazer parte do ciclo de vida do *software* médico com o objetivo de colaborar com a redução da lacuna entre os requisitos de segurança e *safety* da aplicação. Para validar a proposta desta pesquisa, foi realizado estudo de caso em um *software* para dispositivo médico que monitora os sinais vitais de um paciente e realiza alertas relacionados a seu estado de saúde.

Palavras-chaves: Segurança da Informação; Segurança de *Software; Hackers;* Ciclo de Vida do *Software* Médico; Riscos de Segurança.

ABSTRACT

LIFECYCLE BEST PRACTICES TO INCREASE SOFTWARE SECURITY INTO MEDICAL DEVICES

Author: DIOGO DE CARVALHO RISPOLI
Supervisor: Dra. Lourdes Mattos Brasil
Post-Graduation Program in Biomedical Engineering
Brasília, June of 2013.

Despite the widespread use of information technology in the medical field, little attention is given to security software that performs or collects patient information on medical devices. Security software flaws in these devices can lead to problems ranging from the unauthorized disclosure of medical information to the patient injuries or deaths. Aiming to improve the quality and increase safety of this type of equipment, this work proposes a methodology for including security activities in software life cycle to medical devices, especially in its construction and testing phases. This methodology was developed based on the evaluation of standards for implementation of medical devices software and correlation techniques to build secure software. With the adoption of secure development techniques, activities such as abuse cases, risk analysis, code review and penetration testing become part of medical software life cycle in order to collaborate with the reduction of the gap between security and safety requirements of the application. In order to validate the proposal of this research was conducted a case study in medical device software that monitors the vital signs of a patient and performs alerts related to their health status.

Key-words: Information Security; Security Software; Hackers; Medical Software Lifecycle; Security Risks.

SUMÁRIO

LISTA DE TABELAS

LISTA DE QUADROS

LISTA DE FIGURAS

LISTA DE SÍMBOLOS, NOMENCLATURAS E ABREVIAÇÕES

ABNT – Associação Brasileira de Normas Técnicas

ACM – *Association for Computing Machinery*

ANVISA - Agência Nacional de Vigilância Sanitária

BSIMM – *Business Security In Maturity Model*

CBEB XXII – XXII Congresso Brasileiro de Engenharia Biomédica

CRM – Conselho Regional de Medicina

FDA – Food and Drug Administration

IBICT – Instituto Brasileiro de Informações em Ciência e Tecnologia

IDE – *Integrated Development Environment*

IEC – *International Electrotechnical Commission*

IEEE – *Institute of Electrical and Electronics Engineers*

ISO – *International Organization for Standardization*

ITA – Instituto Tecnológico da Aeronáutica

NDA – *Non-Disclosure Agreement*

SAMM – *Software Assurance Maturity Model*

OWASP – *Open Web Application Security Project*

PMBOK – *Project Management Body of Knowledge*

PUC-RIO – Pontificia Universidade Catolica do Rio de Janeiro

QSR – *Quality Systems Regulation*

SDLC – *Software Development Lifecycle*

SDK - *Software Development Kit*

SQUARE – *System Quality Requirements Engineering*

SQL – *Structured Query Language*

TI – Tecnologia da Informação

UFSM – Universidade Federal de Santa Maria

UnB – Universidade de Brasília

USP – Universidade de São Paulo

1 INTRODUÇÃO

1.1 CONTEXTUALIZAÇÃO E FORMULAÇÃO DO PROBLEMA

Em um cenário de constante evolução tecnológica, a busca por soluções que atendam a área médica é uma constante. Atualmente é possível encontrar *software* disponível gratuitamente pela *internet* que coletam informações sobre a saúde dos indivíduos e que podem ser instalados em dispositivos móveis, como *smartphones* e *tablets* (FRASER, 2011).

Existe, inclusive, uma preocupação dos consumidores e profissionais da área de saúde a respeito dos sistemas de informação inteligentes que podem ser utilizados fora do consultório médico. Os participantes da pesquisa (FRASER, 2011) informaram o desejo de dividir os dados coletados por dispositivos de saúde ubíquos com as pessoas ou instituições responsáveis por acompanhar a sua saúde, mas temem que estas informações possam ser acessadas de forma indiscriminada por indivíduos ou grupos que desejem manipular inadequadamente estas informações.

Esta realidade abre portas para que *hackers* explorem e promovam usurpações como o ataque a uma bomba de insulina promovido, documentado e apresentado em (RADCLIFFE, 2011). Baseado em uma técnica simples, que alia conhecimentos de programação e eletrônica básica, o *hacker* empreende um ataque a uma bomba de insulina de seu próprio uso com o objetivo de demonstrar a perspectiva inocente com que estes equipamentos são construídos. Como resultado final, ele consegue aplicar uma dose letal de insulina quebrando a segurança de autenticação imposta pela comunicação sem fio do equipamento.

Vale lembrar que este não foi o primeiro caso de ataque documentado a equipamentos médicos. Em 2008, uma equipe norte-americana de estudiosos publicou um artigo em que mostra um ataque a um marca-passo/desfibrilador, que expunha também falhas de segurança relacionadas à conexão sem fio do equipamento (HALPERIN *et al*., 2008).

Evidentemente, é premente a adoção de padrões e ações que garantam a segurança do *software* presente nos dispositivos médicos. Apesar de atuais, as normas voltadas para a área médica não disponibilizam modelos em que seja possível avaliar e mitigar os

problemas associados a vulnerabilidades comuns de segurança que podem aparecer nos equipamentos médicos, especialmente as falhas relacionadas ao *software*.

Normas de fato como a IEC 62304 (2006) que tratam do desenvolvimento do *software* para equipamentos médicos, apesar de atuais, não lidam com o problema da universalização deste tipo de serviço. Assim como a norma ISO 27799 (2008) que lida com problemas relacionados à segurança dos sistemas de informação médica, mas não endereça soluções relacionadas ao desenvolvimento de *software* seguro.

A exploração de vulnerabilidades em equipamentos médicos podem levar a morte ou lesões graves, fraudes, revelação não autorizada de informação, usurpação, entre outros ataques. Por este motivo, deve-se garantir que requisitos de segurança da informação (integridade, confidencialidade, disponibilidade e não repúdio da informação coletada) sejam cumpridos, bem como assegurar que vulnerabilidades de *software* não sejam inseridas nestes dispositivos durante o seu ciclo de desenvolvimento.

Este trabalho aborda uma visão para a melhoria da segurança de aplicações médicas a partir da avaliação de riscos à segurança da informação e da correlação de requisitos para segurança do *software* (e suas técnicas de mitigação) com os padrões relacionados à segurança no suporte à vida. Os resultados desta correlação são boas práticas propostas para o ciclo de vida do *software* presente em dispositivos médicos.

1.2 OBJETIVOS

1.2.1 Objetivo geral

Reunir padrões, normas técnicas, metodologias e boas práticas de desenvolvimento seguro em um compêndio para minimizar riscos e falhas associados à segurança de *software* para dispositivos médicos.

1.2.2 Objetivos específicos

Este trabalho possui objetivos que podem ser detalhados seguindo aspectos relacionados à engenharia de *software* e a engenharia biomédica:

Quanto à engenharia de *software*, este trabalho se propõe a:

- Estabelecer boas práticas de segurança que serão inseridas no ciclo de vida do desenvolvimento de *software*;

- Propor uma metodologia de construção para aplicações que contemple aspectos relacionados à segurança de *software* para garantir *safety;*
- Demonstrar a execução de testes estáticos e dinâmicos de aplicações para dispositivos médicos.

Quanto à engenharia biomédica, este trabalho se propõe a:

- Avaliar os diversos aspectos de construção de aplicações para equipamentos médicos, incluindo as preocupações relacionadas à segurança do *software*;
- Instruir-se nas normas já existentes para regulação, construção, verificação e validação de *software* para dispositivos médicos;
- Pesquisar aspectos de confidencialidade, integridade e disponibilidade da informação coletada, processada e distribuída pelos dispositivos médicos;
- Correlacionar aspectos de segurança e *safety* na construção do *software* para dispositivos médicos;
- Propor modelos para o mapeamento e mitigação de falhas e vulnerabilidades presentes em equipamentos médicos devido a problemas na construção do seu *software* associado.

1.3 REVISÃO DA LITERATURA

A base da bibliografia referenciada neste trabalho levou em conta a busca por artigos, teses, monografias e livros nas fontes especializadas a seguir: UnB (Universidade de Brasília), USP (Universidade de São Paulo), IBICT (Instituto Brasileiro de Informações em Ciência e Tecnologia), IEEE (*Institute of Electrical and Electronics Engineers*), ACM (*Association for Computing Machinery*) e PubMed. Além disso, foram realizadas pesquisas nas bases de dados dos organismos de normatização ISO (*International Organization for Standardization*), IEC (*International Electrotechnical Commission)* e ABNT (Associação Brasileira de Normas Técnicas).

Foram realizadas pesquisas nas bases bibliográficas da UnB, USP e IBICT, que são instituições de renome no Brasil. Para esta busca foram utilizados os termos “security medical software”, “medical software development security”, “medical software safety”, “segurança de software”, “safety software”.

Na UnB foi encontrado 1 resultado para o termos “segurança de software”, porém este não se mostrou relevante. Todos os outros termos pesquisados não apresentaram nenhum resultado.

Na USP, foram encontrados 21 resultados para o termo “segurança de software”, porém apenas 1 resultado tinha relação correlata com o tema abordado, apesar da baixa relevância. Para o termo “safety software” foram encontrados 12 resultados, entretanto apenas um apresentou baixa relevância. Os outros termos não retornaram resultados. Cabe salientar que para os dois termos pesquisados, o trabalho encontrado é o de Ivanoff (2006) que trata da importância da utilização de processos de desenvolvimento para a construção de *software* seguro em organizações virtuais. Apesar de não ser um trabalho voltado para a área de engenharia biomédica, trata de aspectos correlatos ao desenvolvimento de *software* seguro.

No IBICT foram encontrados 322 resultados para o termo “segurança de software”, dos quais 3 trabalhos mostraram alguma relevância, apesar de pertencem a temas correlatos. No primeiro trabalho encontrado, Batista (2007) discorre sobre os aspectos relacionados a métricas de segurança de *software* com o objetivo de mensurar o impacto das vulnerabilidades em programas de computador. No segundo trabalho encontrado, Filho (2008) fala sobre a importância da adoção de mecanismos de segurança *(safety)* na construção de *software* para sistemas críticos na área da aviação. Para o ultimo resultado relevante, Fontoura (2011) apresenta aspectos de desenvolvimento de *software* confiável a partir da adoção de padrões e modelos de maturidade de segurança de *software.*

Ainda no IBICT, a consulta ao termo “safety software” retornou 153 resultados, porém apenas um representava trabalho em área correlata, mas também com baixa relevância. Abreu (2008) relata em seu trabalho métodos para a aplicação de modelos de melhoria e avaliação de processos de construção de *software* para sistemas críticos de segurança. Não foram apresentados resultados para os outros termos pesquisados.

Foram realizadas pesquisas em bases internacionais do IEEE, ACM e PubMed para os termos a seguir: “security medical software”, “medical software development security”, “medical software safety”. A intenção era buscar por resultados relevantes dentro da área médica e como as bases do IEEE e ACM reúnem trabalhos para diversas áreas do conhecimento, o termo “medical” foi utilizado em todos os parâmetros de pesquisa.

No IEEE, a pesquisa pelo termo “security medical software” retornou 694 e após uma filtragem na pesquisa foram encontrados 4 resultados relevantes. Para o termo “medical software development security” foram encontrados 153 resultados, com 4 resultados relevantes mas semelhantes aos já encontrados. Dentre os resultados encontrados, Halperin (2007) trata de técnicas para a segurança e privacidade em dispositivos médicos implantáveis, mas não relaciona estes dois quesitos a questões de *software*. Já Halperin (2008) trata do problema de segurança em marca-passos a partir de ataques de *software* e propõe algumas mitigações baseadas em ajustes de *hardware*. Howard (2006) trata de processos para a revisão de código fonte do ponto de vista da segurança de *software*. Por ultimo, McGraw (2004) descreve atividades no ciclo de vida do *software* para a melhoria da segurança.

Finalizando as consultas nas bases do IEEE, para o termo “medical software safety”, a pesquisa retornou 478 resultados, com apenas 2 resultados relevantes. No primeiro, Miniati (2010) mostra o que evitar para falhar no gerenciamento da manutenção e do plano de segurança (*safety)* do *software* médico. Por ultimo, Rakitin (2006) mostra como lidar com *software* defeituoso a luz da norma IEC 62304 (2006) e os problemas de *safety* que isto pode ocasionar. Todos os trabalhos encontrados na pesquisa as bases do IEEE são os mais relevantes dentro da linha de pesquisa estudada.

Na ACM, foram encontrados nas pesquisas pelo termo “security medical software” 7450 resultados. A partir do refinamento da pesquisa para trabalhos relacionados a área médica, foi selecionados 2 artigos relevantes. No primeiro trabalho, Gollakota (2011) aborda técnicas de defesa para a conexão sem fio de dispositivos médicos implantáveis. No outro trabalho, Denning (2010) também trata de aspectos de segurança e protocolos para proteger a conexão *wireless* de dispositivos como marca-passos e desfibriladores implatáveis. Para a pesquisa ao termo “medical software development security” foram encontrados 5573 resultados e após o refinamento da pesquisa foi encontrado 1 artigo relevante, semelhante a um resultado que já havia aparecido na primeira pesquisa a base da ACM.

Por fim, na base da ACM, a consulta ao termo “medical software safety” retornou 4366 resultados, com apenas um artigo relevante. Neste artigo, Knight (2002) aborda aspectos de dependabilidade de sistemas embarcados, como forma de garantir *safety* do *software.*

No PubMed, foram realizadas as pesquisas para os termos em questão, porém apenas 1 artigo relevante foi encontrado ao buscar pelo termo “security medical software”. Esta consulta retornou 1035 resultados. Fu (2011) aborda em seu artigo perspectivas de melhoria da qualidade do *software* para dispositivos médicos a partir da especificação de requisitos mais adequados para *safety*, segurança e privacidade.

Foram realizadas pesquisas as bases de dados dos órgãos normatizados ISO, IEC e ABNT em busca de normas que são utilizadas no processo de construção e acreditação de equipamentos médicos e o seu *software* associado. Foram encontradas 9 normas que tratam direta ou indiretamente dos processos para a construção dos equipamentos médicos, ciclo de vida do *software* médico e dos processos de validação e verificação das características dos produtos em âmbito internacional. Dentro deste conjunto, as normais mais relevantes para este projeto de pesquisa são a norma IEC 62304 (2006), que trata do ciclo de vida do *software* para dispositivos médicos, processos e atividades para garantia de *safety*, validação e verificação; a norma ISO 27799 (2008), que trata de sistemas de informação seguros para a área médica, uma especialização da norma ABNT ISO IEC 17799 (2005); a norma ISO 14971 (2007) que trata do gerenciamento de riscos para dispositivos médicos; e a norma ISO/IEC 21827 (2008) que aborda aspectos relativos a modelos de maturidade para a construção de sistemas seguros.

Como forma de complementar e enriquecer a consulta à bibliografia, para os aspectos relacionados à segurança de *software* de um modo geral, realizou-se consulta em mecanismos de pesquisa na *internet* referente a autores renomados que abordam estes assuntos em diversos livros. Desta forma, foram utilizadas referências como Paul (2012) que trata de aspectos relativos a ciclo de vida do *software* seguro, McGraw (2006) que relaciona atividades no ciclo de vida do *software* para a elevação de sua segurança, Long (2012) que aborda aspectos relativos à busca de vulnerabilidades em código fonte, e, finalmente, Stuttard (2012) e Bacudio (2011), que tratam de técnicas de exploração de vulnerabilidades em *software* a partir de testes de penetração.

Apesar das pesquisas apresentarem trabalhos relevantes, nenhum deles aborda a mitigação dos problemas de segurança que podem estar presentes no *software* para equipamentos médicos. Também não foram encontrados trabalhos que propõem técnicas de mitigação e avaliação de vulnerabilidades no *software* para dispositivos médicos a partir da aplicação de atividades no ciclo de vida do *software*.

1.4 ORGANIZAÇÃO DO TRABALHO

Este trabalho está organizado em seis capítulos, incluindo este capítulo.

No capítulo dois, é apresentada uma visão geral do referencial teórico, objetivando a compreensão dos conceitos e padrões utilizados pelas organizações para a construção de *software* para equipamentos médicos e os aspectos teóricos para a inclusão de boas práticas de segurança no ciclo de vida do *software* para dispositivos médicos.

O capítulo três detalha a metodologia utilizada no estudo.

O capítulo quatro descreve os resultados obtidos e o processo de trabalho realizado para a aplicação das boas práticas de segurança no *software* no ciclo de vida do estudo de caso.

O capítulo cinco discute os pontos de maior importância envolvendo o tema deste estudo e apresenta as conclusões finais do trabalho.

Por fim, o capítulo seis apresenta os trabalhos futuros que podem ser desenvolvidos a partir das ideias apresentadas neste documento.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 SEGURANÇA DA INFORMAÇÃO

A informação é um ativo que necessita ser adequadamente protegido, pois é essencial para os negócios de uma organização. A informação pode existir de diversas formas. Especialmente no que tange os meios eletrônicos a informação está exposta a um grande número ameaças e vulnerabilidades, visto que cada vez mais os ambientes de negócio estão interconectados (ABNT ISO IEC 17799, 2005).

A segurança da informação é preservação das propriedades de confidencialidade, integridade e disponibilidade da informação. Adicionalmente, outras propriedades como autenticidade, responsabilidade, não repúdio e confiabilidade podem estar envolvidas (ABNT ISO IEC 17799, 2005). A confidencialidade é a garantia de que a informação está protegida contra revelação não autorizada. A integridade é a propriedade pela qual a informação não será modificada indevidamente e a disponibilidade é a percepção de que a informação estará disponível sempre que necessário. A relação de hierarquia entre a segurança da informação e os seus pilares, pode ser observada na Figura 1.

Figura 1: Relação entre a segurança da informação e os seus pilares (Modificado de ABNT ISO IEC 17799, 2005).

Ela é necessária para proteger a informação dos diversos tipos de ameaça e para auxiliar a continuidade das atividades relativas ao negócio. Para obtê-la, é necessário implementar um conjunto de controles adequados, incluindo politicas, processos, procedimentos e funções de *software* e *hardware*. A segurança da informação e seus processos exigem monitoração, análise e melhoria constante para garantir a segurança da

organização e os seus objetivos de negócio (ABNT ISO IEC 17799, 2005). Na área da saúde, a privacidade dos sujeitos depende da manutenção da confidencialidade das informações pessoais de sua saúde (ISO 27799, 2008).

Para os sistemas de informação médicos, existem diversos tipos de informação as quais a confidencialidade, integridade e disponibilidade precisam ser protegidas (ISO 27799, 2008):

- Dados pessoais de saúde;
- Dados derivados de informações pessoais de saúde, incluindo dados coletados e identificados a partir de pseudônimos;
- Dados de pesquisa e estatísticos, incluindo dados anônimos derivados de informações pessoais de saúde;
- Dados sobre profissionais de saúde, equipes e voluntários;
- Trilhas de auditoria, produzidos por sistemas de informação médicos que contém dados pessoais de saúde, dados pseudônimos derivados de informações pessoais de saúde ou que contem dados sobre ações dos usuários no que diz respeito ao uso de informação pessoal de saúde;
- Dados de segurança para os sistemas de informação de saúde, incluindo dados de controle de acesso e outros dados relacionados a configurações de segurança dos sistemas de informação de saúde.

A extensão do que deve ser protegido do ponto de vista da segurança da informação depende da natureza da informação, do tipo de uso da informação bem como dos riscos que trarão a sua exposição (ISO 27799, 2008).

2.1.1 Segurança de *software*

A segurança de *software (software security* [1]*)* é um campo que surgiu a partir da necessidade de desenvolvedores, arquitetos e cientistas da computação de lidar com a construção de *software* mais seguro (MCGRAW, 2004).

De forma prática, o *software* resistente a invasão é o que reduz a probabilidade de um ataque bem sucedido e mitiga a extensão do dano se o ataque ocorrer. Para que o *software*

[1] As palavras, do inglês, *security* e *safety* tem a mesma tradução para o português, que é segurança. Apenas como forma de diferenciar os conceitos de *safety software* e *software security*, a palavra *safety* será mantida em inglês e remeterá aos conceitos de *safety software* e suas técnicas correlatas.

seja seguro e resistente a invasão, ele deve levar em conta conceitos de segurança de *software* (PAUL, 2011). Outra visão é: o *software* seguro é aquele que não pode ser intencionalmente subvertido ou forçado a falhar. Ou seja, ele se mantem correto e previsível mesmo após o esforço intencional de comprometer seu funcionamento (GOERTZEL *et al*, 2007).

O *software* seguro é desenhado, implementado, configurado e suportado de forma que é possível estabelecer que ele continue operando corretamente na presença dos principais ataques, resistindo à exploração de falhas ou outras fraquezas no *software* pelo atacante, ou ainda tolerando os erros e falhas que podem resultar desta exploração, bem como isole, contenha e limite o dano resultante de falhas causadas por qualquer defeito acionado por ataques que o *software* não consiga resistir ou tolerar e se recupere o mais rápido possível destas falhas (GOERTZEL *et al*, 2007).

As seguintes propriedades caracterizam a segurança do *software* (GOERTZEL *et al*, 2007):

- Falhas exploráveis e outras fraquezas são evitadas por desenvolvedores bem intencionados;
- A probabilidade de que desenvolvedores mal intencionados implantem falhas exploráveis ou lógicas maliciosas no *software* é baixa ou nula;
- O *software* será resistente, tolerante ou resiliente a ataques.

Para desenvolver *software* seguro, é importante incorporar conceitos de segurança nas fases de requisito, projeto, codificação, liberação e descarte do ciclo de vida do *software*. Incorporar conceitos de segurança é uma necessidade básica que precisa ser abordada e que não pode ser ignorada na construção das aplicações (PAUL, 2011).

2.1.2 Safety software

A disciplina de *safety software*[2] está preocupada em garantir a confiança dos sistemas computacionais a partir de um julgamento quantitativo das suas propriedades. Este conceito está ligado a dependabilidade, que é a propriedade que define a capacidade dos

[2] As palavras, do inglês, *security* e *safety* tem a mesma tradução para o português, que é segurança. Apenas como forma de diferenciar os conceitos de *safety software* e *software security*, a palavra *safety* será mantida em inglês e remeterá aos conceitos de *safety software* e suas técnicas correlatas.

sistemas computacionais de prestar um serviço que se pode justificadamente confiar (KNIGHT, 2002).

A noção de dependabilidade pode ser quebrada em seis propriedades fundamentais relacionada aos sistemas computacionais, a seguir (KNIGHT, 2002):

- Confiança: manterá o funcionamento correto quanto estiver em uso;
- Disponibilidade: estará operacional quando necessário;
- *Safety*: sua operação não trará perigo ao usuário ou operador;
- Confidencialidade: não haverá revelação não autorizada da informação;
- Integridade: não haverá modificação não autorizada da informação;
- Manutenibilidade: facilidade de modificação do *software* com objetivo de corrigir defeitos, se adequar a novos requisitos ou se ajustar a um ambiente novo.

É importante observar os requisitos de segurança não estão explicitamente incluídos nesta lista, porém eles estão cobertos pelas propriedades disponibilidade, confidencialidade e integridade (KNIGHT, 2002).

2.1.3 Falhas de segurança em *software* e *hardware*

Com sua gama intrínseca de componentes, os equipamentos médicos estão sujeitos a partes com problemas. Dentro destas diversas partes, identificar e quantificar no *software* os potenciais efeitos de falhas e defeitos é complexo. Vários dispositivos compartilham componentes eletrônicos e mecânicos em comum e seus processos de busca, correção e documentação de falhas já estão bem consolidados (RAKITIN, 2006).

Por outro lado, geralmente o *software* é construído sob demanda e para ser utilizado em dispositivos específicos. Com raras exceções, não existem mecanismos efetivos para estabelecer e mapear falhas ou problemas no *software*. Como consequência, recai sobre os fabricantes a responsabilidade de comprovar que o *software* para equipamentos médicos são seguros e efetivos (RAKITIN, 2006).

2.2 CICLO DE VIDA DO DESENVOLVIMENTO DE SOFTWARE

Um modelo de ciclo de vida organiza as atividades de desenvolvimento de *software* e provê um *framework* para monitorar e controlar o seu projeto de construção e operação. Sem a adoção de um modelo é difícil endereçar em que momento o desenvolvimento e a

validação do projeto se encontram e até mesmo como e em que situações os pontos de controle devem ser aplicados (RAKITIN, 2006; MCGRAW 2006).

O ciclo de vida faz parte dos controles do projeto e estes controles são necessários para reduzir a probabilidade de inserção de defeitos no dispositivo médico (VOGEL, 2011). Então, para mitigar o problema das vulnerabilidades dentro do *software* médico, é essencial indicar atividades de segurança de *software* a serem aplicadas entre as fases do ciclo de vida das aplicações. Estas atividades estão relacionadas à identificação, construção e validação de técnicas que impossibilitem a exploração de vulnerabilidades na operação do *software*.

Com a adoção de padrões de construção de *software* seguro dentro do seu ciclo de vida, a partir de atividades que garantam a identificação, avaliação, tratamento, aplicação e validação de controles de segurança da informação, espera-se o aumento da qualidade e diminuição máxima das possibilidades de ataque dentro das aplicações médicas. O levantamento e o desenvolvimento de *checklists* com os controles a serem aplicados pode auxiliar a incorporação de práticas de codificação defensiva ao longo da construção das aplicações médicas.

O tratamento dos aspectos relacionados à segurança do *software* não necessariamente representa aumento do custo no seu ciclo de vida de desenvolvimento, tendo em vista que corrigir problemas e falhas desta natureza custam mais depois da aplicação pronta e em produção (MCGRAW, 2006).

A adoção de técnicas de segurança nos dispositivos médicos é esperada porque o *software* médico pode ser executado em diversos tipos de plataformas, dentre elas computadores pessoais e dispositivos móveis computacionais. Toda essa diversidade de plataformas pode trazer riscos de lesões e do vazamento de informações da relação médico-paciente, dado que toda informação coletada e transmitida por estes dispositivos é sensível, confidencial ou até pode mudar o funcionamento dos equipamentos.

2.3 GARANTIA DOS REQUISITOS DE QUALIDADE

Apesar das QSRs (*Quality Systems Regulations*) não estabelecerem um modelo específico de ciclo de vida para o *software* médico, as normas regulatórias colocam que a adoção de um modelo é importante e que este deve conter pelos menos as fases de planejamento da

qualidade, definição de requisitos, especificação do projeto de *software*, construção (ou codificação), teste, instalação, operação e suporte, manutenção e retirada de circulação (RAKITIN, 2006; MCGRAW 2006).

2.3.1 IEC 62304:2006

Existem diversos padrões, conforme é possível observar na Figura 2, que tratam direta (como a norma IEC 62304) ou indiretamente do ciclo de desenvolvimento de *software* e os seus riscos associados (HALL, 2010). Entretanto, existe uma carência de metodologias que abordem aspectos ligados à mitigação de vulnerabilidades exploráveis em *software*.

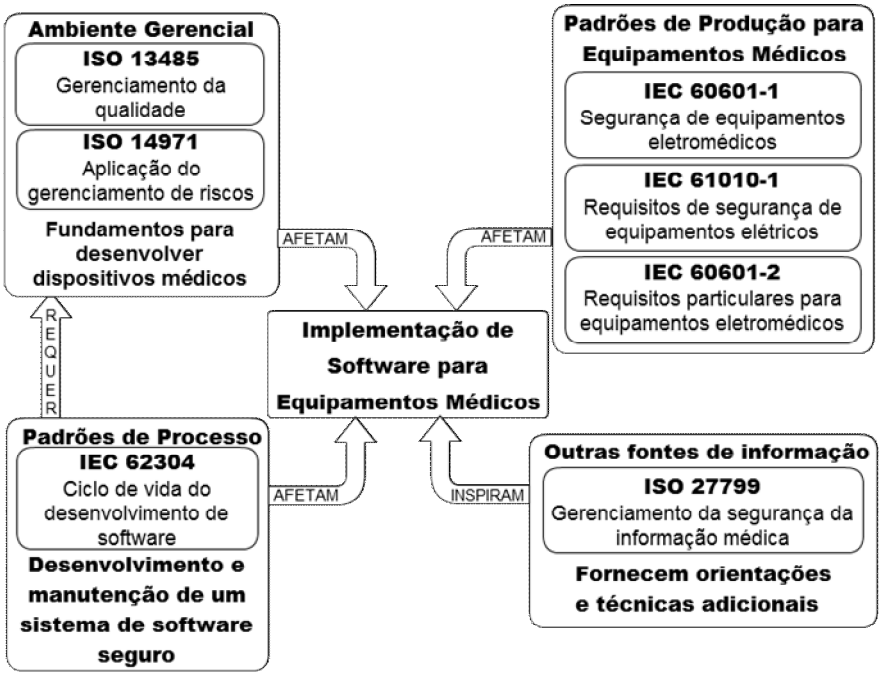

Figura 2: Norma IEC 63304 e seus relacionamentos com outros padrões (HALL, 2010).

As atividades e preocupações descritas na norma IEC 62304 (2006) não estão relacionadas com modelos de desenvolvimento de *software* específicos. Elas podem ser aplicadas ao ciclo de vida do *software* independente da metodologia de desenvolvimento adotada.

2.3.2 ISO 27799:2008

A norma ISO 27799 (2008), em sua preocupação relacionada aos sistemas de informação médica, não trata ou endereça soluções relacionadas ao processo de construção de *software* seguro. Apesar disto, coloca requisitos para a operação de sistemas de informação seguros

como autenticação, autorização, responsabilização, utilização de criptografia, transporte seguro da informação e proteção contra códigos móveis que são aspectos relevantes em que o *software* é o ator principal ou atua como coadjuvante importante (ISO 27799, 2008).

2.3.3 ISO 14971:2009

A norma ISO 14971 (2009) especifica o processo pelo qual um fabricante pode avaliar os riscos associados em produtos desenvolvidos para a área médica, bem como estimar e controlar estes riscos, monitorando a eficácia dos controles implementados a partir da criação de sistemas de gerenciamento de riscos (ISO 14971, 2009). Uma correlação entre as etapas de monitoramento de riscos relativos à segurança do *software* pode ser observada na Figura 3.

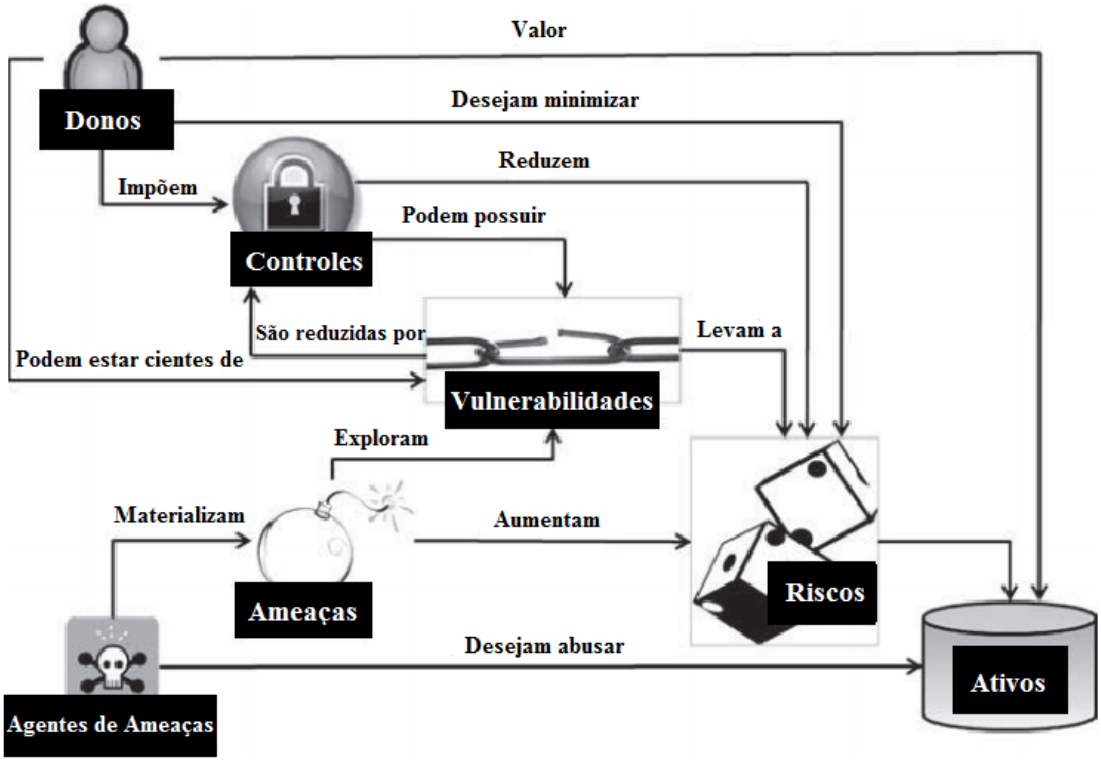

Figura 3: Gerenciamento dos riscos de segurança (Adaptado de PAUL, 2011).

Com relação à perspectiva do risco, uma forma de associar questões que estão aparentemente desconectadas é observar como aspectos de segurança de *software* estão ligados a construção de dispositivos médicos seguros. Sob o olhar do *software*, os riscos são caminhos através da aplicação por onde atacantes podem prejudicar o negócio ou as organizações (OWASP, 2010), como pode ser visualizado na Figura 4.

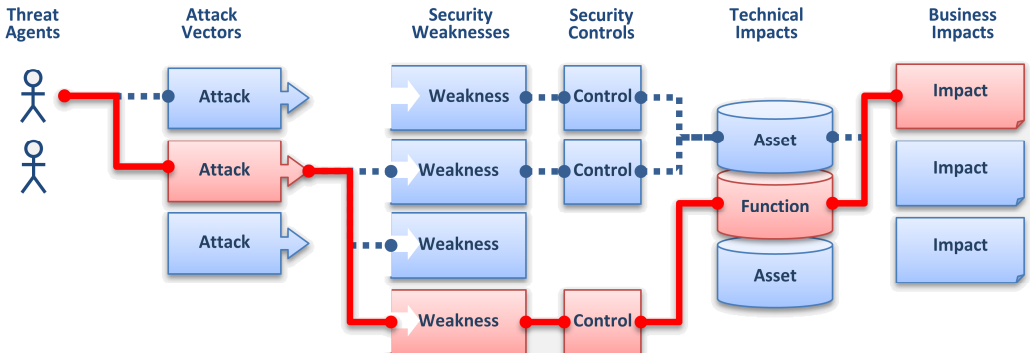

Figura 4: Riscos de Segurança da Aplicação (OWASP, 2010).

Observando-se o mesmo aspecto pela perspectiva do equipamento, o risco (ou nível de preocupação) é uma estimativa da severidade da lesão que um equipamento pode permitir ou infligir. Esta estimativa leva em conta que estas lesões podem acontecer de forma direta ou indireta, em um paciente ou no operador do equipamento, a partir de falhas do dispositivo, falhas de projeto, ou simplesmente, em virtude do emprego do dispositivo para seu uso pretendido (FDA, 2005).

Objetivamente, estas duas visões se aproximam, pois o risco está diretamente ligado a falhas do *software* ou fraquezas em seus mecanismos de controle. Sua consequência natural é a subversão e danos de diversas naturezas (primariamente danos à vida, mas também financeiro e da imagem corporativa dos fabricantes) provocados por pessoas mal intencionadas.

2.3.4 Normatização por Órgãos Reguladores

É importante também que a responsabilidade de avaliar os aspectos de segurança dentro do *software* produzido e utilizado pelos dispositivos médicos faça parte dos programas de verificação e validação adotados pelos organismos designados para tal. Órgãos como a ANVISA (Agência Nacional de Vigilância Sanitária) no Brasil podem instituir dentro do seu modelo de validação e verificação de qualidade de *software* análises estáticas e dinâmicas, empreendidas de forma manual ou automatizada, com o objetivo de analisar se o *software* que acompanha os dispositivos médicos é robusto na capacidade de resistir a ataques promovidos por *hackers*.

2.3.5 Validação e verificação

Observando os processos de garantia da qualidade empregados para as aplicações médicas, atualmente, os aspectos de validação e verificação estão somente preocupados com requisitos funcionais do *software* (VOGEL, 2011).

2.4 SOFTWARE ASSURANCE

O termo *software assurance* até recentemente era mais relacionado a duas propriedades de *software*: qualidade e confiança. Nos últimos anos esse termo passou a ser adotado para expressar a ideia de garantia da segurança de *software*, quando comparado garantia de segurança da informação, a qual é expressa pelo termo garantia da informação (GOERTZEL, 2007).

Todas as definições de *software assurance* transmitem a ideia de que este deve fornecer um nível razoável de confiança justificável de que o *software* funcionará corretamente, de forma previsível e de maneira consistente com seus requisitos documentados (GOERTZEL, 2007). Portanto, a função do *software* não deve ser comprometida, mesmo durante ataques diretos ou sabotagem.

A melhor definição para o termo garantia de *software* seria que esta é a base para obter a confiança de que o *software* irá exibir, consistentemente, todas as propriedades necessárias para assegurar que o *software*, em operação, continuará a operar apesar da presença de falhas intencionais (GOERTZEL, 2007). Em termos práticos, tal *software* deve ser capaz de resistir e conter danos, além de recuperar seu nível normal de operação o quanto antes for possível e ser capaz de resistir e tolerar qualquer ataque.

2.5 ATIVIDADES NO CICLO DE VIDA DO SOFTWARE

Um modo para melhorar a segurança e a qualidade do *software* é realizar atividades, ou boas práticas, de segurança através do ciclo de vida do *software*. Estas atividades, que são na realidade boas práticas de segurança de *software*, são responsáveis por lidar com as preocupações de segurança e precisam ser aplicadas dentro das fases do ciclo de vida da aplicação em vez de fazê-lo apenas na fase de requisitos, como sugerido pela norma IEC 62304 (2006). Uma correlação entre as fases do ciclo de vida e as atividades podem ser visualizadas na Figura 5 e foram descritas, de forma geral, para projetos de *software* por McGraw (2006).

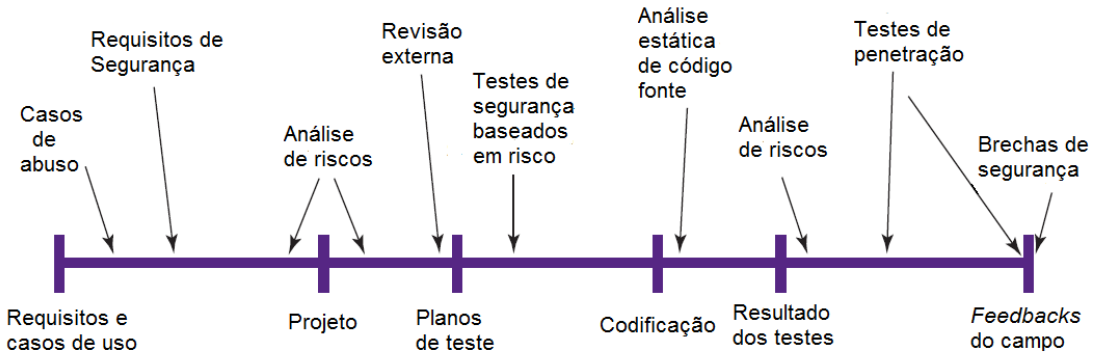

Figura 5: Atividades de segurança no ciclo de vida do *software* (MCGRAW, 2004).

A seguir, uma breve descrição das atividades a serem realizadas ao longo do ciclo de vida (MCGRAW, 2006):

- Casos de abuso: Construir casos de abuso é relevante para realizar uma relação entre os problemas e a análise de risco. É importante observar neste momento se algum padrão de ataque se encaixa no sistema ou nos requisitos do *software*. Este é um bom momento para modelar cenários de vulnerabilidades que podem ser exploradas nas fases de revisão de código ou teste de penetração.
- Requisitos de segurança: Os requisitos de segurança precisam cobrir os requisitos funcionais e de segurança, casos de abuso levantas de padrões de ataque. Nesta fase toda a necessidade de segurança do *software* precisa ser mapeada para garantir sua correta implementação. Um bom exemplo de requisito de segurança esta relacionado com o uso correto de criptografia para proteger dados críticos.
- Análise de risco arquitetural: Complementa a análise de riscos orientada pela ISO 14971 (2007). Esta análise está relacionada com o levantamento dos riscos de segurança que podem estar presentes na arquitetura da aplicação que será construída e é uma pequena parte do processo de gerenciamento de risco que todo fabricante necessita aplicar para garantir aderência à referida norma, de acordo com (IEC 62304, 2006).
- Teste de segurança baseado em riscos: A estratégia de testes da aplicação precisa cobrir pelo menos dois tópicos principais, que são os testes dos requisitos de segurança com técnicas de teste para requisitos funcionais e testes de segurança baseados em riscos, levantados pelos casos de abuso e pela avaliação dos padrões de ataque.

- Revisão de código fonte: Após a fase de codificação e antes da fase de testes, a análise de código fonte é uma boa atividade para garantir que os requisitos de segurança foram bem implementados e que as vulnerabilidades listadas na análise de casos de abuso não estão presentes no *software.* A revisão de código pode ser automática e manual e cada estratégia tem prós e contras. Ferramentas automatizadas não cobrem todos os cenários, por este motivo a análise manual é sempre necessária (LONG *et al*., 2012).
- Testes de penetração: Este é um conjunto de técnicas e ferramentas utilizadas em conjunto para testar dinamicamente um *software* ou sistema contra falhas de projeto ou vulnerabilidades. Esta atividade é importante para garantir que a aplicação ou sua infraestrutura não possuem nenhum problema potencial que podem ser explorados de uma forma particular para alterar o comportamento da aplicação em tempo de execução (MICROSOFT, 2008).
- Operação segura: É importante responsabilizar as atividades do usuário no momento da utilização do sistema de *software*. Ainda mais importante é manter estes dados de forma correta e protegida, para garantir que o atacante ou atividades de ataque possam ser rastreadas após qualquer tentativa de ataque, bem sucedida ou não.

Cabe ressaltar que as atividades de segurança descritas podem ser aplicadas a qualquer tipo de ciclo de vida de *software,* bem como independem do modelo de desenvolvimento de *software* adotado pelos desenvolvedores. Estas atividades não tem nenhuma ligação direta com o modelo de desenvolvimento de *software,* apesar da sua eficácia na melhoria da qualidade das aplicações.

Para apoiar a implementação das boas práticas de *software*, pode-se estabelecer a criação de um Grupo de Segurança de *Software*, com pessoas que possuem experiência real em desenvolvimento de aplicações. Esta é uma maneira de garantir que estas atividades serão acompanhadas por pessoas com conhecimento técnico especializado no desenvolvimento de *software* seguro (BSIMM, 2012).

2.6 IMPLEMENTAÇÃO DE TÉCNICAS DE MITIGAÇÃO DE VULNERABILIDADES EM SOFTWARE

As vulnerabilidades em *software* são problemas inerentes ao seu processo de construção. Dentre outros, estas falhas podem estar relacionadas com defeitos comuns que podem

aparecer a partir do levantamento incorreto dos requisitos funcionais e não funcionais do *software*, especialmente aqueles relacionados a questões de segurança; lacunas no planejamento do projeto; falhas no processo de garantia da qualidade e testes bem com a falta de controles adequados dentro do ciclo de desenvolvimento de *software*. O levantamento de riscos e problemas durante a fase de projeto da aplicação é fundamental para garantir que as principais vulnerabilidades serão mitigadas adequadamente durante a fase de codificação.

Mesmo que todos os controles, verificações e processos relativos a segurança sejam realizados e aplicados, ainda não há garantia que outras vulnerabilidades irão aparecer. As vulnerabilidades podem estar relacionadas com o ambiente onde a aplicação é executada, bem como elas ainda podem não ter sido descobertas, documentadas e/ou exploradas em grande escala. As atividades relativas à segurança de *software* estão preocupadas em aumentar a resistência da aplicação a ataques, pois não existe *software* sem vulnerabilidades. Existem aplicações resilientes que dificultam a exposição e exploração de falhas de segurança.

Então, é necessário tratar e listar interações com o ciclo de vida do *software* que orientam como realizar testes de penetração ou auditorias, levantar requisitos não funcionais para implantação e operação segura de aplicações, incluir a análise de risco de vulnerabilidades no projeto do *software*, proteger as aplicações contra falhas de injeção de comando e *buffer overflow*, tratar corretamente erros e exceções e responsabilizar gravando registros sanitizados (*i.e.,* após remoção de informações sensíveis) da atividade dos usuários na operação do equipamento.

Também é importante projetar mecanismos eficientes de autenticação, gerenciamento de sessão segura, autorização do usuário autenticado, criptografia para o transporte e armazenamento seguro das informações coletadas e dos registros dos pacientes com o objetivo de aumentar a garantia da segurança e da qualidade dos sistemas de informação de saúde e suas aplicações correlatas (que tem como ativos os dispositivos e seu *software* associado).

Falhas de *software* são consequências naturais da análise, projeto, e construção das aplicações, porém uma visão inocente pode diminuir a capacidade crítica de observar que

falhas são aspectos que vão além dos defeitos (*bugs*) e que o risco relacionado às vulnerabilidades de segurança não pode ser ignorado.

2.7 REQUISITOS DE SEGURANÇA

Os requisitos de segurança aumentam capacidade da aplicação de estar menos vulnerável a possíveis tentativas de ataques e invasões por terceiros. Os requisitos de segurança estão ligados a três atributos de qualidade de *software* (PAUL, 2011):

- Confiança;
- Resiliência;
- Recuperabilidade.

Devido a sua importância, pois através deles é que são endereçadas as principais dificuldades de segurança que uma aplicação deve lidar, os requisitos de segurança precisam ser considerados durante todas as fases de construção e desenvolvimento de um *software* (PAUL, 2011).

2.7.1 Fontes de requisitos de segurança

As fontes de requisitos de segurança podem ser tanto internas como externas. As fontes internas se caracterizam por serem organizacionais, requisitos que as organizações devem cumprir inevitavelmente. As fontes internas de requisitos de segurança incluem as políticas, as diretrizes, os padrões e as práticas da instituição. O usuário final do negócio também é outra fonte interna de onde podem ser coletados os requisitos de segurança (PAUL, 2011).

As fontes externas de requisitos de segurança podem ser classificadas em regulamentações, iniciativas de *compliance* e requisitos geográficos. Devem ser atribuídos pesos iguais para os dois tipos de fonte, independente de serem internas ou externas (PAUL, 2011).

Os *stakeholders* (empresários, usuários finais e clientes) possuem papel importante na determinação dos requisitos de segurança de *software* e devem ser envolvidos ativamente no processo de elicitação dos requisitos. Os empresários são responsáveis por determinar qual o nível de risco aceitável, pois são os donos do risco em última instância. Eles devem

auxiliar o os times de desenvolvimento de *software* a avaliar o risco e serem proativos ao dizer o quê é realmente importante para os processos decisórios (PAUL, 2011).

Os donos de negócios devem ser orientados quanto à importância e os conceitos de segurança de *software*, pois isto garantirá que eles não atribuirão uma baixa prioridade aos requisitos de segurança de *software* ou tratá-los como sem importância. Além disso, dar apoio a grupos de operações e de segurança da informação é vital às partes interessadas, e também asseguram que o *software* que está sendo construído é confiável, resiliente e recuperável (PAUL, 2011).

2.7.2 Tipos de requisitos de segurança

Os requisitos de segurança devem ser definidos explicitamente e devem ser corresponder aos objetivos e metas de segurança da organização. Quando os requisitos são apropriadamente definidos e documentados é possível mensurar os objetivos e metas de segurança do projeto, facilitando a implementação e liberação do *software*.

Os requisitos de segurança de *software* podem também estar relacionados com aspectos de regulação e atenção a normas nacionais e internacionais. A relação de alguns dos tipos de requisitos de segurança de *software* pode ser observada a seguir (PAUL, 2011):

- Confidencialidade;
- Integridade;
- Disponibilidade;
- Autenticação;
- Autorização;
- Auditoria;
- Gerenciamento de sessão;
- Gerenciamento de erros e exceções;
- Gerenciamento de parâmetros de configuração;
- Sequenciamento e tempo;
- Arquivamento;
- Internacionais;
- Ambiente de produção;

- Aquisição;
- Antipirataria.

Figura 6: Tipos de requisitos de segurança de *software* (PAUL, 2011).

Na fase de levantamento de requisitos do ciclo de desenvolvimento de *software*, a equipe de segurança de *software* é necessária apenas para identificar quais requisitos são aplicáveis no contexto da organização e para qual funcionalidade do *software* eles devem ser aplicados. Detalhes de como estes requisitos serão implementados devem ser decididos durante o projeto e desenvolvimento do *software* (PAUL, 2011). Exemplos de requisitos de segurança de *software* são apresentados no Anexo 1.

2.7.3 SQUARE

O SQUARE *(System Quality Requirements Engineering)* é uma metodologia desenvolvida na Universidade de Carnegie Mellon para auxiliar organizações a inserirem segurança desde as fases iniciais da produção do ciclo de vida de *software*. Essa metodologia consiste em nove passos que ao final resultam em requisitos de segurança categorizados e priorizados (MEAD, 2005).

O objetivo em longo prazo do SQUARE é integrar considerações de segurança nas primeiras fases do ciclo de vida de desenvolvimento. Além disso, SQUARE provou ser útil em documentar e analisar os aspectos de segurança de *software* que já estão em operação e tem necessidade de direcionar melhorias e modificações futuras nessas aplicações (MEAD, 2005).

A metodologia é mais efetiva quando conduzida por um time de analistas de requisitos em conjunto com os *stakeholders* do projeto. O SQUARE pode ser decomposta em nove passos, conforme é possível observar na Tabela 1 (MEAD, 2005).

Tabela 1 – Nove passos para execução do SQUARE (MEAD, 2005)

	Passo	**Entrada**	**Técnica**	**Particiantes**	**Saída**
1	**Concordar com as definições**	Definição candidata do IEEE e outros padrões.	Entrevistas estruturadas, grupo foco.	*Stakeholders*, equipe de requisitos.	Definições acordadas.
2	**Identificar ativos e objetivos de segurança**	Definições, objetivos candidatos, condutores de negócios, politicas e procedimentos, exemplos.	Sessões de trabalho facilitadas, pesquisas, entrevistas.	*Stakeholders*, analista de requisitos.	Ativos e objetivos.
3	**Desenvolver artefatos para suportar as definições de requisito de segurança**	Artefatos potenciais (e.g. cenários, modelos, formas, caso de abuso, entre outros).	Sessão de trabalho.	Analista de requisitos.	Artefatos necessários, caso de abuso, cenários, modelos, modelos e formas.
4	**Realizar avaliação de risco**	Caso abuso, cenários, objetivos de segurança.	Método de avaliação de risco, análise de risco antecipada contra tolerância de risco organizacional, incluindo análise de ameaças.	Analista de requisitos, *stakeholders*, *expert* em risco.	Avaliação de resultado de risco.
5	**Selecionar elicitações técnicas**	Objetivos, definições, técnicas dos candidatos, experiência dos *stakeholders*, estilo organizacional, cultura, nível de segurança	Sessão de trabalho.	Engenharia de requisitos.	Técnicas de elicitações selecionadas.

		necessário, análise de custo-benefício, entre outros.			
6	**Elicitar requisitos de segurança**	Artefato, risco, avaliação de resultados, técnicas selecionadas.	*Joint aplication development* (JAD), entrevistas, pesquisas, análise de modelo de base, *checklist*, lista de tipos de requisitos reutilizáveis, revisão de documentos.	*Stakeholders* facilitados por analistas de requisitos.	Corte inicial no requisito de segurança.
7	**Categorizar requisitos pelo nível (sistema, *software*, outros) e quando forem requisitos ou outro tipo de restrições**	Requisitos iniciais, arquitetura, avaliação dos resultados.	Sessão de trabalho utilizando categorias padronizadas.	Analista de requisitos, outros especialistas necessários.	Requisitos categorizados.
8	**Priorizar requisitos**	Categorizar requisitos e riscos.	Métodos priorizados como triagem e ganha-ganha.	*Stakeholders* facilitados por analistas de requisitos.	Requisitos prioritários.
9	**Inspecionar requisitos**	Requisitos priorizados, utilizar técnica de inspeção formal.	Método de inspeção, revisões por pares.	Equipe de inspeção.	Requisitos selecionados inicialmente, documentação do processo de tomadas de decisões e análise racional.

2.8 AVALIAÇÃO DE ADERÊNCIA

A avaliação de aderência ao processo de desenvolvimento de *software* seguro é importante para verificar a distância que o *software* analisado e seu ciclo de desenvolvimento estão de um processo de desenvolvimento que tem preocupação com questões de segurança.

Esta análise tem como escopo o *software,* seu ciclo de vida e possíveis atividades já realizadas para observar a resistência da aplicação a ataques e suas mitigações. Para isto, devem ser avaliados também a documentação da aplicação, a sua metodologia de desenvolvimento, bem como seus requisitos e aspectos de funcionamento.

2.9 AVALIAÇÃO DE VULNERABILIDADE EM SOFTWARE

As vulnerabilidades em *software* podem ser avaliadas a partir de testes estáticos ou dinâmicos. Os testes estáticos estão relacionados à verificação de vulnerabilidades exploráveis que podem aparecer no código fonte da aplicação. Já os testes dinâmicos estão relacionados com a busca por problemas com o *software* em execução.

Os testes estáticos podem ser realizados por ferramentas ou manualmente por desenvolvedores experientes. As ferramentas podem auxiliar a busca por problemas de segurança comuns no *software*, causados principalmente por falhas ou descuidos na fase de codificação (MCGRAW, 2006). Para uma análise arquitetural do *software,* bem como na busca por falhas relacionadas à lógica de negócio da aplicação, é necessário realizar uma revisão de código fonte manual. Esta revisão manual é mais cuidadosa e mais lenta do que a análise feita por ferramentas automatizadas, porém tem a capacidade de verificar problemas mais elaborados que não podem ser verificado por ferramentas.

Os testes dinâmicos estão relacionados à utilização de ferramentas e técnicas para exploração de falhas e vulnerabilidades com o *software* em execução. Da mesma forma do teste estático, as ferramentas para teste dinâmico automatizado não conseguem evidenciar vulnerabilidades mais elaboradas, como falhas nas lógicas de negócio do *software* (PAUL, 2011). Neste caso, podem ser utilizadas ferramentas que tem o intuito de apoiar a utilização de técnicas manuais no processo do teste de penetração. No caso da realização dos testes por equipes internas da organização, pode-se utilizar análise dinâmica também para confirmar as vulnerabilidades encontradas no processo de revisão do código fonte *software* (PAUL, 2011).

Como forma ilustrativa, é possível observar na Figura 7 a relação entre os riscos, os testes de vulnerabilidades, os resultados gerados e a mitigação, que também é um resultado esperado neste processo.

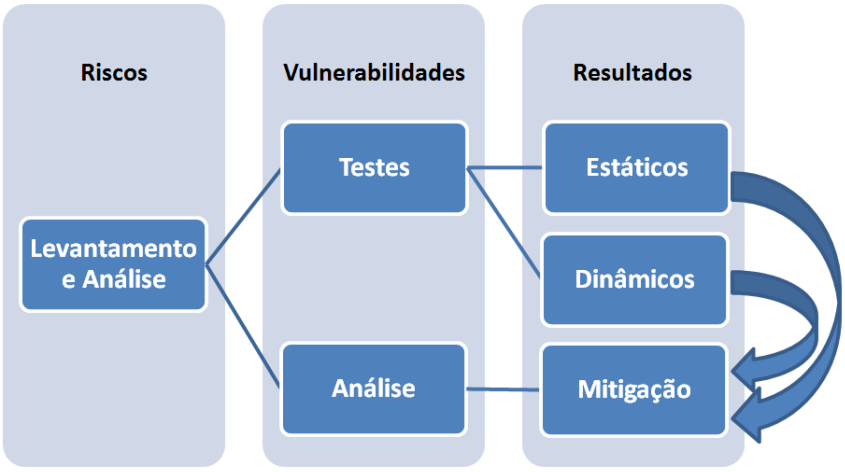

Figura 7: Relação entre riscos, teste de vulnerabilidades e seus resultados.

2.10 MODELOS DE MATURIDADE PARA SEGURANÇA DE SOFTWARE

Um modelo de maturidade é uma representação simplificada do mundo e contêm elementos essenciais para a construção de processos efetivos. Modelos de maturidade focam no melhoramento dos processos em uma organização. Eles contêm os elementos essenciais para a construção de processos efetivos para uma ou mais disciplinas e descreve um caminho de melhoria evolucionaria que vai desde processos *ad-hoc*, seguindo por processos imaturos, disciplinados e finalizando nos processos maduros com qualidade e efetividade melhorada (SEI, 2010).

Os modelos de maturidade proveem orientação para ser utilizada no desenvolvimento de processos efetivos, porém, não são processos ou descrições de processos. O processo adotado em uma organização depende de inúmeros fatores, incluindo seus domínios de aplicações e a estrutura e o tamanho da organização. Em particular, as áreas de processo de um modelo de maturidade não mapeiam um a um com os processos usados na organização como um todo (SEI, 2010).

2.10.1 OpenSAMM

O OpenSAMM (*Open Software Assurance Maturity Model)* é um *framework* aberto para auxiliar organizações a formular e implementar suas estratégias de segurança de *software* que são adaptados aos riscos específicos enfrentados pela organização. Os recursos oferecidos pelo OpenSAMM irão ajudar em (CHANDRA, 2009):

- Avaliar as práticas de segurança de *software* existentes na organização;
- Construir e equilibrar o programa de garantia de segurança de *software* em interações bem definidas;
- Demonstrar melhoramentos concretos no programa de garantia de segurança;
- Definir e mensurar atividades relacionadas à segurança por toda organização.

OpenSAMM foi concebida com flexibilidade, de forma que pode ser utilizada por pequenas, médias ou grandes organizações e que façam uso de qualquer estilo de desenvolvimento. Esse modelo pode ser aplicado por toda organização, para uma única linha de negócios ou para um projeto individual (CHANDRA, 2009). Além dessas características, OpenSAMM foi construída pelos princípios a seguir (CHANDRA, 2009):

- Comportamento da organização se modifica lentamente no decorrer do tempo. Um *software* de segurança bem sucedido deve ser especificado em pequenas interações que entreguem ganhos de garantias tangíveis, enquanto trabalha para objetivos de longo prazo.
- Não há uma única receita que funcione para todas organizações. O *framework* de segurança de *software* deve ser flexível e permitir que as organizações formem suas escolhas baseado em sua tolerância de riscos e na direção na qual constrói e usa o *software*.
- Orientação relacionada a atividades seguras devem ser prescritivas – Todos esses passos na construção e avaliação em um programa de garantia deve ser simples, bem definido e mesurável. Esse modelo oferece modelos de roteiros para tipos comuns de organizações.

A fundação do modelo é construído sobre o núcleo das funções do negócio de desenvolvimento de *software* com práticas seguras associadas a cada função. Os tijolos da construção do modelo são os três níveis de maturidade definidos para cada uma das suas dozes práticas de segurança. Eles são definidos por uma vasta variedade de atividades, nas quais a organização pode reduzir os riscos de segurança e aumentar a garantia de qualidade de *software* (CHANDRA, 2009).

2.10.2 BSIMM

O BSIMM *(Building Security In Maturity Model)* é um estudo de iniciativas de segurança *software* existentes. Quantificando as práticas de várias organizações distintas, pode-se

descrever o solo comum compartilhado por muitas, e também as variações que fazem cada uma única (MCGRAW, 2012).

O principal objetivo é ajudar o plano comunitário de *software* seguro a realizar e medir suas próprias iniciativas. O BSIMM não é um guia de como fazer, ele é um reflexo do estado da arte do *software* seguro aplicado as organizações (MCGRAW, 2012).

O trabalho com o modelo BSIMM mostra que mensurar a iniciativa de segurança de uma organização é possível e extremamente útil. As mensurações do BSIMM podem ser utilizadas para planejar, estruturar e executar as evoluções de uma iniciativa de segurança de *software* (MCGRAW, 2012). O BSIMM pode ser usado por alguém responsável por criar e executar iniciativas de segurança de *software* e traz a confiança do conhecimento das melhores práticas sobre *software* seguro para estabelecer um *framework* de segurança de *software* (MCGRAW, 2012).

Durante a criação do modelo BSIMM, foram conduzidas uma série de entrevistas com executivos responsáveis por nove iniciativas de *software* seguro. Nestas entrevistas, foram identificadas atividades em comum às nove iniciativas avaliadas, atividades estas que compõem o *framework* de segurança de *software* do BSIMM (MCGRAW, 2012).

A seguir, foram criados cartões com pontuações que mostravam as atividades que foram utilizadas para cada uma das nove iniciativas avaliadas (MCGRAW, 2012). A fim de validar o trabalho, foi solicitado para cada participante revisar o *framework*, as práticas, e os cartões com as pontuações foram criados para suas iniciativas.

As cinquenta e uma organizações participantes foram retiradas de dozes diferentes setores: serviços financeiros, vendedores independentes de *software*, empresas de tecnologia, computação nas nuvens, mídia, segurança, telecomunicações, seguros, energia, varejo, saúde e provedores de *internet* (MCGRAW, 2012).

Na média, os participantes tinham prática com segurança de *software* por aproximadamente 6 (seis) anos. Todas as 51 (cinquenta e uma) empresas concordaram que o sucesso de seus programas ocorreu devido ao seu grupo de segurança de *software* (MCGRAW, 2012).

O BSIMM foi criado para instruir como iniciativas de *software* seguro fornecem recursos para organizações que procuram constituir ou melhorar suas próprias iniciativas de *software* seguro (MCGRAW, 2012).

No geral, qualquer iniciativa de *software* seguro é criada com alguns objetivos em mente. O BSIMM é apropriado se os objetivos dos seu negócio para segurança de *software* incluem (MCGRAW, 2012):

- Decisões de gerenciamento de risco informadas;
- Clareza no que é a coisa certa a fazer para todos envolvidos em segurança de *software*;
- Melhoria da qualidade de código.

2.11 ENGENHARIA DE SOFTWARE EXPERIMENTAL

A experimentação, como centro do processo científico, oferece um modo sistemático, disciplinado, controlado e computável para verificação de teorias. Considerando a engenharia de *software*, o desafio é determinar se esta pode ser chamada de ciência ou engenharia. Considerando o processo de criação do produto, a engenharia de *software* assume naturalmente as características explicitas de produção ou engenharia. Por outro lado, os aspectos relacionados a melhoria continua da qualidade do processo e do produto colocam a engenharia de *software* no contexto de ciência (TRAVASSOS, 2002).

Neste modelo, metodologias específicas são necessárias para ajudar a estabelecer bases de engenharia e ciência para a engenharia de *software*. Existem diversas métodos relevantes para a condução de estudos na área da engenharia de *software*. Cada método tem como foco os aspectos específicos ao tipo de experimentação proposto.

A abordagem da engenharia de *software* experimental sugere um modelo, desenvolve o método de avaliação qualitativo e/ou quantitativo, aplica um experimento, mede e analisa, avalia o modelo e repete o processo sempre que necessário. Este processo se inicia com o levantamento de um modelo novo, que pode ser baseado em um modelo já existente, e tenta estudar o efeito do processo ou produto sugerido pelo novo modelo (TRAVASSOS, 2002).

Esta metodologia é útil porque considera a proposição e avaliação do modelo com os estudos experimentais. Porém, é importante observar que os experimentos de fato não

realizam prova com certeza absoluta. Eles apenas verificam a previsão teórica de encontro à realidade (TRAVASSOS, 2002).

Entre objetivos relacionados à execução de experimentos em engenharia de *software* estão a caracterização, avaliação, previsão, controle e melhoria a respeito de produtos, processos, recursos, modelos, teorias, entre outros. A experimentação pode auxiliar a construção de uma base de conhecimento confiável e apoiar a redução das incertezas sobre as teorias, ferramentas e metodologias abordadas, bem como acelerar o processo eliminando abordagens inúteis e suposições errôneas (TRAVASSOS, 2002).

Os principais elementos relacionados aos experimentos são as variáveis, os objetos, os participantes, o contexto do experimento, hipóteses e o tipo de projeto do experimento. Existem dois tipos de variáveis do experimento, que são as variáveis independentes e dependentes. As variáveis independentes estão relacionadas com a entrada do processo de experimentação e apresentam a causa que afeta o resultado do processo de experimentação. Da mesma forma, as variáveis dependentes referem-se às saídas do experimento e apresentam aos efeitos causados pelos fatores do experimento. O valor relacionado a estas variáveis dependentes são chamados também de resultados (TRAVASSOS, 2002). O efeito da correlação destes conceitos pode ser visualizado na Figura 8.

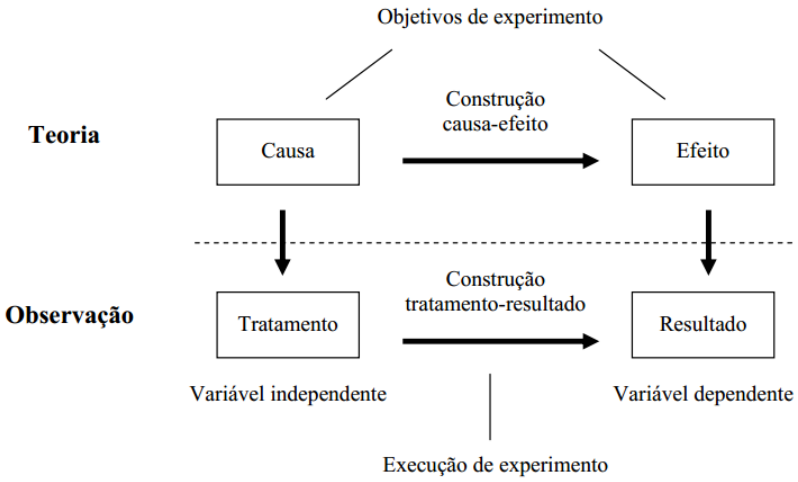

Figura 8: Conceitos de um experimento (WOHLIN, 2000).

No caso da engenharia de *software*, o processo de medição utiliza as próprias métricas do *software*. Estas métricas são responsáveis por medir as características do processo de desenvolvimento de *software*. Elas têm como objetivo principal aumentar a compreensão do processo e do produto, controlando-os ao definir antecipadamente as atividades

corretivas e identificando as possíveis áreas de melhoria. A validade dos resultados do experimento está ligada a capacidade de generalização do experimento em relação a populações mais amplas do que as abordadas ao longo do processo de experimentação (WOHLIN, 2000).

O tipo de experimento mais adequado para uma população concreta está relacionado aos objetivos do estudo, às propriedades do processo de software adotado ou aos resultados finais esperados. Porém, em engenharia de *software*, a classificação dos experimentos está sempre relacionada aos conceitos das estratégias empíricas. Em linhas gerais, são encontradas na literatura três principais estratégias experimentais (TRAVASSOS, 2002):

- *Survey*, que é uma investigação realizada em retrospectiva e faz sua coleta de informação baseada em questionários;
- Estudos de caso, que são utilizados para monitorar os projetos, atividades e atribuições. Eles visam observar um atributo especifico e estabelecer a relação entre atributos diferentes e possui baixo nível de controle;
- O experimento em si, que geralmente é realizado em laboratório e oferece um maior nível de controle. Tem como objetivo manipular algumas variáveis experimentais e manter outras fixas, verificando o efeito sob o resultado.

Pode-se observar no Quadro 1 uma comparação das principais estratégias experimentais.

Fator	***Survey***	**Estudo de caso**	**Experimento**
Controle da Execução	Nenhum	Nenhum	Possui
Controle da Medição	Nenhum	Possui	Possui
Controle da Investigação	Baixo	Médio	Elevado
Facilidade da repetição	Elevado	Baixo	Elevado
Custo	Baixo	Médio	Elevado

Quadro 1: Comparação das estratégias experimentais empíricas.
Fonte: TRAVASSOS, 2002.

3 METODOLOGIA

3.1 GARANTIA DA SEGURANÇA DE *SOFTWARE* NO SEU CICLO DE VIDA

A proposta deste estudo é propor uma metodologia para a construção de *software* seguro para equipamentos médicos através da utilização de boas práticas de segurança de *software* ao longo do seu ciclo de vida. Esta metodologia está baseada na inclusão de atividades de levantamento de requisitos de segurança, análise dos riscos, elicitação de casos de abuso, mitigação dos riscos no código fonte, análise estática de código fonte, testes de segurança baseados em riscos, teste de penetração e estabelecimento de parâmetros para configuração e instalação segura das aplicações.

As atividades, ou boas práticas, podem ser utilizadas em conjunto, seguindo uma sequência que acompanha o ciclo de vida do *software* ou adotadas separadamente, da maneira que fizer maior sentido para a organização. O modelo adotado para a execução das boas práticas sugeridas deve levar em consideração também questões como o orçamento e o prazo do projeto de *software*, além de considerar que a mesma atividade pode ser realizada por diversas abordagens diferentes. Assim, ao longo do desenvolvimento das atividades dentro do ciclo de vida do *software* devem ser escolhidas as abordagens de execução que se mostrarem mais adequadas para o cenário proposto.

É importante estabelecer que não existe obrigatoriedade da adoção de todas as atividades em todas as iterações do ciclo de vida. Por exemplo, em uma iteração para correção de defeitos estéticos, como o alinhamento de campos no formulário de uma página HTML ou um ajuste no texto de ajuda do *software* provavelmente não justificam um teste de penetração completo na aplicação ou no fluxo modificado, a não ser que ocorra modificação ou inclusão de funcionalidades.

Por outro lado, existem organizações que realizam determinadas atividades de acordo com uma necessidade específica. Em situações onde a infraestrutura de execução da aplicação sofre alguma modificação, como a troca de um servidor de aplicação ou com a atualização do sistema operacional não se faz necessária uma nova revisão do código-fonte da aplicação, porém o teste de penetração passa a ter grande valor.

Variáveis como a linguagem de programação, ambiente de execução (plataforma de *hardware* e sistema operacional), complexidade e tamanho do projeto de *software* também impactam na execução destas boas práticas de construção de *software* seguro. Por este motivo, não é possível endereçar de forma geral uma disciplina tão abrangente que não necessite de ajustes no momento da aplicação prática.

3.2 PROPOSTA

Dado o cenário onde o *software* passa pela primeira iteração do ciclo de vida, em fases de projeto e construção, dificilmente será possível suprimir alguma boa prática. Neste caso, é desejável que o *software* tenha incorporado nesta iteração todas as atividades propostas neste trabalho. Na Figura 9 é possível observar a relação entre as atividades propostas e o ciclo de vida do *software* para equipamentos médicos, em sua aplicação completa. Uma descrição sobre as atividades e sua aplicação no ciclo de vida se apresenta nos parágrafos a seguir.

Primeiramente, os requisitos de segurança da aplicação são definidos em conjunto com o levantamento de requisitos do *software*. Estes requisitos podem estar relacionados com aspectos de autenticação e autorização, proteção da informação, responsabilização, entre outros. Cada aplicação tem os seus requisitos definidos de forma individualizada, pois é importante pesar a relação custo-benefício para sua implementação. O levantamento dos requisitos de segurança acontece na fase inicial do projeto, ou na construção de novas funcionalidades, para que possam ser avaliados e levados em consideração durante todas as fases seguintes do ciclo de desenvolvimento do *software*.

Os riscos, na perspectiva de segurança, estão diretamente relacionados com falhas de *software* ou vulnerabilidades. Por exemplo, o risco para a vulnerabilidade de injeção SQL é o vazamento de informações e para autenticação e autorização são usuário ilegítimos acessando e explorando a aplicação.

A análise de risco deve observar normas para o levantamento e o gerenciamento de riscos, como a ISO 14971 (2009), e sua relação com os riscos de segurança de aplicações. Riscos de segurança estão também relacionados com o tipo de *software* a ser desenvolvido. Por exemplo, para aplicações *standalone*¸ problemas típicos relacionados ao roubo do *cookie* de sessão não podem ser considerados como risco, pois estes são problemas inerentes a aplicações *web*.

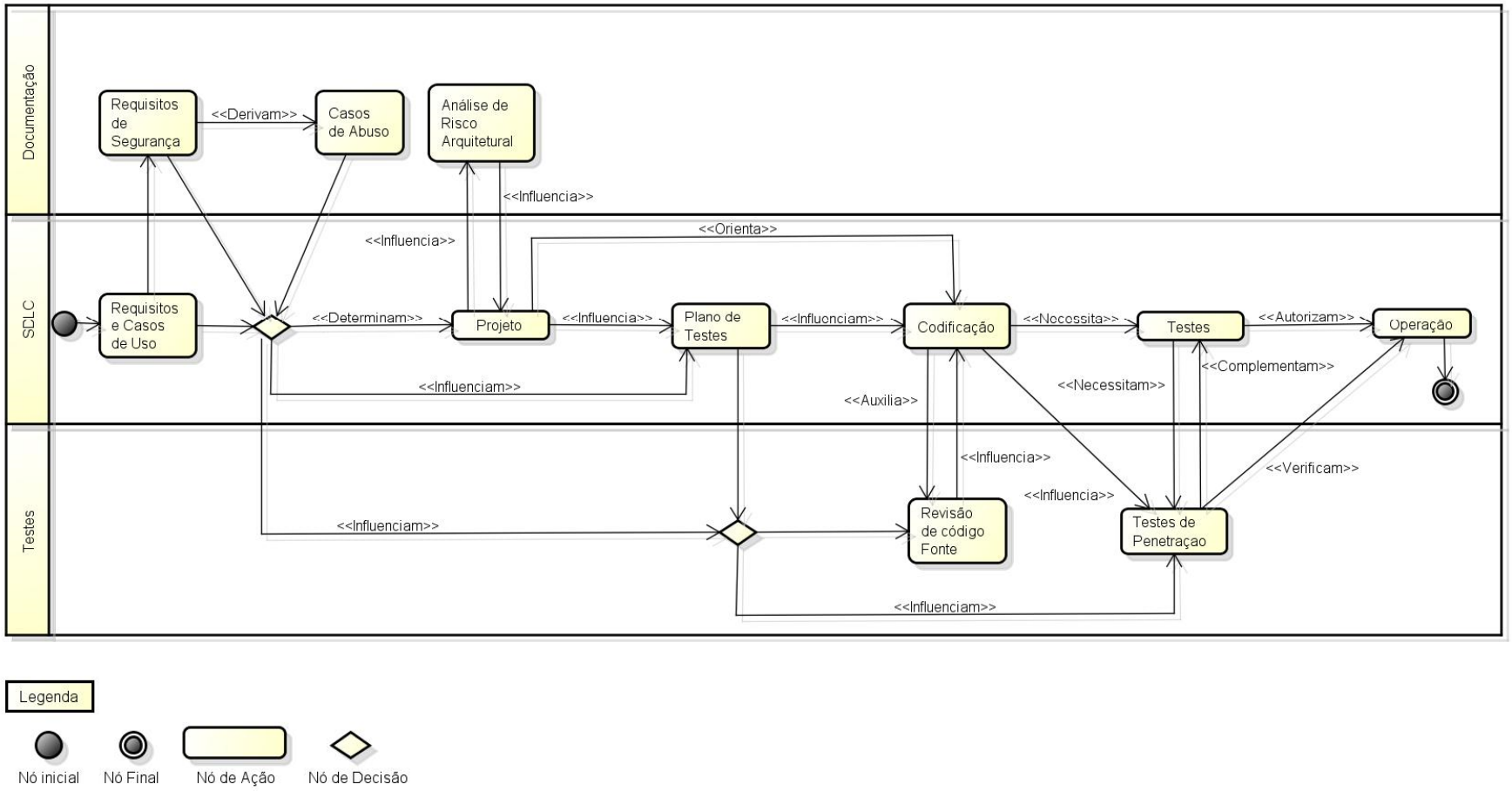

Figura 9: Atividades de segurança aplicadas ao ciclo de vida do *software* para equipamentos médicos.

Após a análise e o levantamento dos riscos, devem ser confeccionados os casos de abuso. Os casos de abuso ligam vulnerabilidades conhecidas aos riscos definidos na fase anterior. Um caso de abuso é definido a partir de quais explorações a aplicação pode sofrer tendo em vista os riscos que ela possui. Por exemplo, uma aplicação pode possuir um risco de expor informações sobre a condição física de um paciente. Então, o caso de abuso vinculado a este risco pode estar relacionado a tentativas de injeção de comandos, como a injeção SQL, com o objetivo de retirar estas informações sensíveis da base de dado de forma fraudulenta.

Casos de abuso estão intimamente relacionados com vulnerabilidades e falhas de *software*. Injeção SQL (*Structured Query Language*), seguindo com o exemplo anterior, pode revelar problemas na validação dos dados inseridos na aplicação. A possibilidade de injeção de comandos é uma vulnerabilidade comum em *software* e precisa ser mitigada na fase de construção da aplicação. Existem casos de abuso que endereçam problemas relativos a falhas no projeto de *software*. Problemas de autenticação e autorização são exemplos de falhas que podem surgir por deficiência na definição dos riscos e requisitos de segurança de aplicação.

A fase de codificação é o momento onde uma parte das questões levantadas nas etapas anteriores deve ser avaliada e mitigada. Apesar da equipe de segurança não participar diretamente da execução desta atividade, a sua participação nas etapas anteriores garante uma maior facilidade da equipe de desenvolvimento de endereçar e resolver os problemas de segurança no código-fonte da aplicação.

Após a fase de codificação, é o momento de verificar se os requisitos de segurança foram implementados de forma correta. Esta atividade consiste, pelo menos, nos seguintes elementos: rastrear pontos de entrada de dados controlados pelo usuário e revisar o código fonte responsável por processá-los procurando por evidências que garantam que não existam vulnerabilidades relacionadas aos riscos no código fonte; bem como examinar, buscando por padrões conhecidos relacionados a vulnerabilidades comuns.

Neste momento, um processo de revisão de código-fonte deve ser iniciado. Esta revisão, que é uma estratégia de controle importante, pode ser realizada manualmente, linha-a-linha, com o objetivo de verificar se todos os requisitos de segurança foram implementados, bem como todos os riscos levantados foram mitigados. Essa verificação

manual exige muita experiência em desenvolvimento de *software*. Por este motivo o revisor deve possuir perfil de desenvolvedor.

A revisão automatizada de código-fonte também é realizada nesta mesma fase e é útil para agilizar o processo de revisão manual, pois esta funciona a partir da busca de assinaturas no código-fonte que podem indicar que os riscos comuns não foram mitigados de forma correta. Apesar do ganho de velocidade que as ferramentas trazem para a avaliação do código-fonte é importante entender que as ferramentas não fazem todo o trabalho. A revisão manual é sempre necessária pois é nesta abordagem que falhas relacionadas as características de negócio e a lógica de execução do *software* são encontradas.

Na fase de codificação estes riscos devem ser mitigados para garantir que não existirão caminhos na aplicação para exploração dessas vulnerabilidades. Os testes de segurança baseados em risco estão relacionados com os riscos levantados. Nesta etapa, a aplicação sofrerá uma série de testes baseados nos casos de abuso além do seu processo de homologação das funcionalidades face aos casos de uso.

Nesta etapa, os testadores utilizarão os casos de abuso levantados e tentarão burlar as mitigações para os riscos de segurança a partir da operação normal da aplicação. Problemas encontrados nesta fase precisam ser reportados para a equipe de desenvolvimento, que farão as correções necessárias antes do teste de penetração. Estes testes não precisam ser realizados por testadores com habilidades de atacante (*hackers*), tendo em vista que a aplicação passará por diversos ataques na próxima etapa.

Para confirmar que os problemas encontrados na revisão de código fonte podem ser realmente explorados, um teste de penetração precisar ser realizado. Existem, pelo menos, três fases envolvidas no teste de penetração: preparação do teste, execução do teste e a análise do teste, como pode ser visualizado na Figura 10.

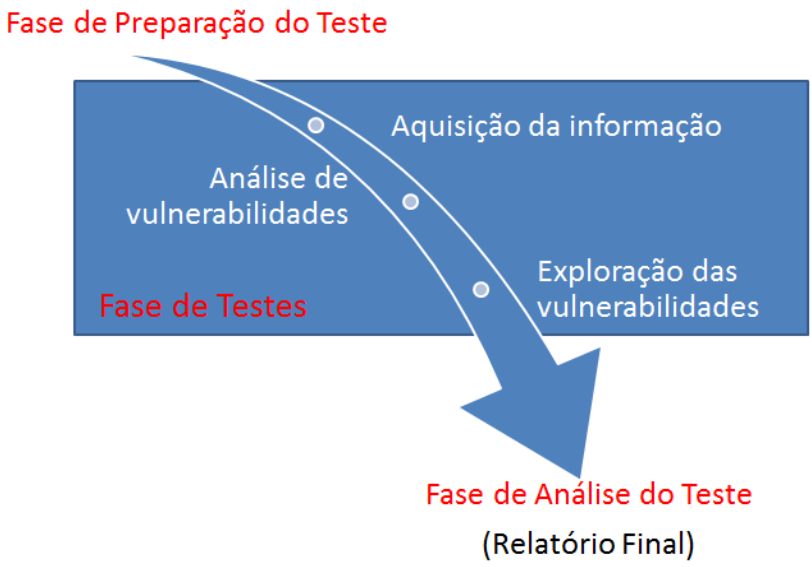

Figura 10: Fases do teste de penetração (Modificado de BACUDIO et al, 2011).

A primeira fase esta relacionada com a definição do escopo, objetivo, horários e a duração do teste. Todos os acordos legais precisam ser organizados nesta fase. A segunda fase é considerada o cerne do processo do teste de penetração. Esta fase envolve a coleta de informação sobre o aplicativo, a análise e a exploração das vulnerabilidades. Os resultados são investigados e analisados na última fase. O relatório final gerado precisa ser compreensível e sistemático.

Neste teste, a aplicação será escrutinada por atacantes profissionais ou *hackers* éticos. Nesta fase, a aplicação será avaliada em um ambiente de execução próximo do ambiente real onde ficará em funcionamento. O teste de penetração possui diversas fases, como reconhecimento, mapeamento, verificação e exploração das vulnerabilidades e o seu relatório final. Caso sejam encontradas vulnerabilidades nesta fase, os testadores devem encaminhar à equipe de desenvolvimento um relatório com os seus achados e como eles foram explorados. Isto é especialmente útil para que os desenvolvedores consigam realizar os ajustes de forma precisa, antes do novo teste de penetração.

Por ultimo, mas não menos importante, são as preocupações relativas à segurança do ambiente de execução da aplicação. Cada *software* possui requisitos de operação segura intrínsecos e eles devem ser definidos levando-se em conta as melhores práticas relacionadas ao ambiente onde a aplicação será executada.

A operação segura está preocupada com problemas da plataforma que podem ocorrer quando o *software* está funcionado. Exemplos dos requisitos descritos são: proteção dos dados, aspectos práticos de criptografia e uso de canais de comunicação protegidos.

Adicionalmente, para aplicações que já possuem partes desenvolvidas deverão aplicadas verificações de aderência aos padrões de codificação segura. Esta verificação consiste em avaliar quão distante o *software* está da implementação de mecanismos para mitigação de riscos de segurança que podem comprometê-lo e propor uma forma adequada de melhoria ao longo de uma nova fase do seu ciclo de desenvolvimento. Por ser realizada uma nova iteração no ciclo de desenvolvimento de *software* apenas para endereçar problemas de segurança encontrados.

3.3 DELIMITAÇÃO DO ESTUDO

Este estudo, embora possa envolver o uso de *hardware* utilizado como dispositivo médico, está preocupado apenas com as questões de segurança que envolvem o *software* para equipamentos médicos e suas características intrínsecas.

Atividades que não envolvam aspectos de segurança como testes para garantia de qualidade, sejam eles para homologação, validação e verificação em *software*, bem como metodologias específicas de desenvolvimento de *software* não serão abordados neste trabalho, pois, os testes realizados neste trabalho têm como objetivo avaliar questões relativas a falhas e vulnerabilidades de segurança em *software*. Da mesma maneira, como não é necessária a adoção de um modelo específico de desenvolvimento de aplicações para que as técnicas apresentadas neste trabalho possam ser aplicadas, este item não necessita ser avaliado.

3.4 ESTUDO DE CASO

Para o tipo de experimento avaliado neste trabalho, julgou-se adequada a abordagem do estudo de caso, em consonância com as práticas da engenharia de *software* experimental.

3.4.1 *Software* avaliado

O dispositivo medico que teve o seu *software* avaliado é responsável por monitorar os sinais vitais de um paciente e enviar as informações coletadas para um dispositivo *Android*. O sistema, exibido como diagrama na Figura 11 é dividido entre *Body Sensor Network* (Figura 12a), que é um conjunto de sensores que monitoram os sinais vitais, o sensor

Coordinator que coleta as informações dos sensores corporais regularmente e reenvia estes dados para o *smartphone* e o *software* Monitor (Figura 12b), que avalia a condição clinica do paciente de tempos em tempos.

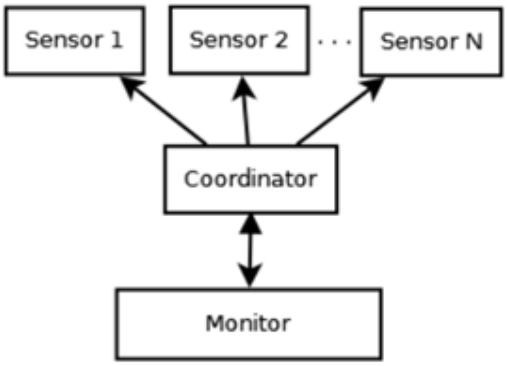

Figura 11: Diagrama do sistema Monitor.

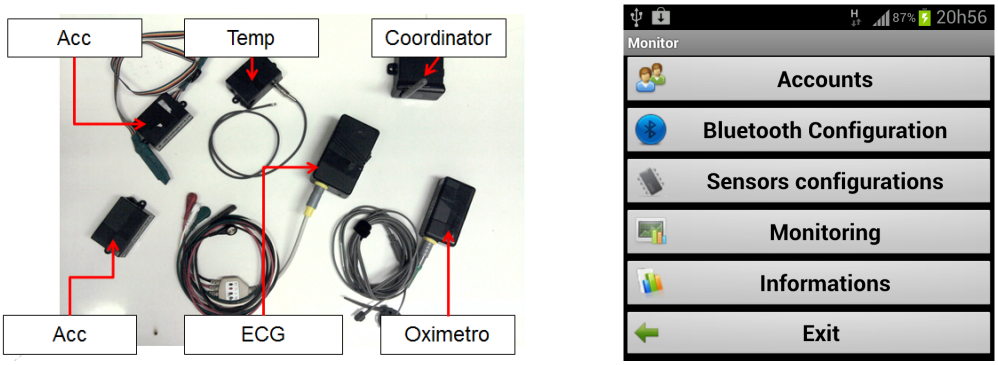

Figura 12: (a) *Body Sensor Network* (esquerda). (b) Interface do *software* Monitor (direita).

O equipamento em questão foi desenvolvido como parte de um projeto de pesquisa do Grupo de Engenharia de *Software* do Departamento de Ciência da Computação da Universidade de Brasília e foi gentilmente cedido para esta avaliação. Um acordo de confidencialidade foi firmado para a avaliação de segurança presente no trabalho e pode ser visualizado no Anexo 2.

Note que a única parte avaliada neste trabalho é o *software* Monitor. Partes mecânicas, sensores e o *hardware* do *smartphone* utilizado não fazem parte desta análise. O *software* Monitor foi desenvolvido na linguagem *Java* para ser executado em dispositivos *Android*.

3.5 COLETA E ANÁLISE DE DADOS

Existem alguns passos a serem seguidos para a realização da avaliação proposta. Estes passos podem estar relacionados com uma ou mais atividades e eles foram realizados para colocar o *software* avaliado dentro de um ciclo de vida seguro.

Todo o método de adoção de boas práticas e avaliação destas atividades de segurança no ciclo de vida do *software* é baseado nos seus processos de desenvolvimento. Mesmo quando os problemas relacionados refletem no ambiente de operação do *software*, ele certamente estará relacionado a falhas durante a fase de levantamento de requisitos ou de ajustes do ambiente de operação da aplicação.

O sensor e seu *software* relacionado são um bom candidato para as avaliações de segurança porque o *software* está em um estagio de desenvolvimento inicial e utiliza uma metodologia não usual de desenvolvimento para dispositivos médicos. É especialmente interessante poder validar se as atividades de segurança realmente se encaixam em uma nova metodologia ou perspectiva de desenvolvimento de *software*.

Sendo assim, toda a informação coletada para esta pesquisa se deu ao longo de uma iteração do ciclo de vida do estudo de caso. Deste modo é possível detectar problemas que podem ser explorados por atacantes na aplicação antes da sua entrada em circulação.

Os dados coletados no *software* Monitor levam ao amadurecimento da postura de segurança da aplicação. A partir da definição de requisitos de segurança, levantamento de casos de abuso e avaliação de riscos arquiteturais da aplicação, incluem-se como insumo para a equipe de arquitetura e desenvolvimento da aplicação propostas de mitigação de falhas que ajudam a reduzir problemas comuns de segurança de *software*. Já na fase de verificação, a equipe de segurança pode apoiar os testes e verificações de qualidade já adotadas pela organização.

Para que seja verificada a redução de vulnerabilidades para um mesmo *software* do ponto de vista histórico, é importante que atividades de segurança aplicadas durante este pesquisa façam parte de toda a iteração futura do ciclo de vida da aplicação.

3.6 RECURSOS TECNOLÓGICOS

Os recursos tecnológicos utilizados neste trabalho compreendem:

- *Software* para o sistema operacional *Android*;
- *Android* SDK (*Software Development Kit*);
- *Eclipse* IDE (*Integrated Development Environment*);
- Distribuição *Linux* para teste de penetração *Backtrack.*
- Ferramentas para busca textual de código-fonte;
- Ferramentas de código aberto para análise de trafego de rede;
- Ferramentas de código aberto para decompilação de aplicativos *Android*;
- Dispositivo *Android*;
- Dispositivo para coleta de sinais vitais com conexão *bluetooth;*
- *Dongle bluetooth.*

3.7 RESTRIÇÕES

Para a realização destas verificações foi firmado um acordo de confidencialidade, chamado também de NDA (*Non-Disclosure Agreement*), com grupo de pesquisa do *software* Monitor com o objetivo de proteger a propriedade intelectual do *software* avaliado e também evitar a revelação indiscriminada das vulnerabilidades encontradas.

O acordo de confidencialidade firmado entre as partes para a análise de segurança do *software* Monitor pode ser encontrado no Anexo 2.

4 RESULTADOS

4.1 ANÁLISE DO SOFTWARE MONITOR

Esta pesquisa vem de encontro à carência de diversos grupos em propor mitigações para os problemas encontrados em *software* para equipamentos médicos, vide a escassez de trabalhos e publicações relativas ao tema de pesquisa proposto.

A análise realizada no *software* verificou, através da execução dos passos descritos na metodologia, eventuais problemas relacionados a falhas de segurança, bem como distância para a aderência ao processo de desenvolvimento de *software* seguro.

Apenas para registrar, as práticas de *safety* listadas na norma IEC 62304 (2006) e em outros padrões não serão ignoradas, apenas será adicionada a elas uma perspectiva de segurança. Estas práticas serão adicionadas ao processo do *software* para aumentar a *safety* e estabelecer segurança. Por exemplo, a análise de risco, casos de abuso e testes baseados em risco já estão presentes em processos relativos à *safety* e este trabalho trará preocupações de segurança a estas atividades.

Com esta análise, foi possível confirmar a importância da adoção de metodologias de construção de *software* seguro dentro de um projeto de equipamento médico. O resultado desta análise revelou problemas que poderiam ser evitados se durante o desenvolvimento a aplicação tivesse utilizado a metodologia descrita neste trabalho no seu ciclo de vida.

4.2 REQUISITOS DE SEGURANÇA

Através da avaliação dos requisitos funcionais da aplicação e sua correlação com as necessidades abordadas pela norma IEC 62304 (2006), foram estabelecidos alguns requisitos de segurança para o *software* Monitor. Estes requisitos complementares foram estabelecidos como parâmetro para o levantamento dos casos de abuso e as revisões de código fonte e teste de penetração.

Os requisitos de segurança de *software* levados em consideração durante a análise estática e dinâmica do *software* monitor são:

- Autenticação segura;
- Transmissão de informação criptografada entre o sensor e o dispositivo *Android*;

- Proteção do arquivo da base de dados local;
- Validação de entrada contra injeção de comando;
- Alteração do código de conexão com o sensor;
- Armazenamento seguro de dados sensíveis;
- Armazenamento seguro de credenciais;
- Registro das atividades do usuário no *software* Monitor.

4.3 CASOS DE ABUSO

A partir do estabelecimento dos requisitos de segurança, foram elicitados 5 (cinco) casos de abuso. Estes casos de abuso estão relacionados com os problemas que a equipe de segurança da organização entende como explorações que a aplicação sofrerá no momento que ela for disponibilizada ao seu usuário final se os seus requisitos de segurança não forem seguidos e/ou implementados adequadamente.

Então, os casos de abuso se baseiam nos requisitos de segurança porque eles são um guia de como verificar se os requisitos de segurança foram implementados. Desta forma, os casos de abuso que foram definidos para o *software* Monitor, com uma breve descrição da exploração que será aplicada, são:

- Captura do tráfego entre o sensor e o dispositivo *Android*: O atacante obtêm as informações trocadas na conexão entre o sensor e o dispositivo *Android* para análise futura e roubo de informação;
- Vazamento de informação no *software* Monitor: O atacante obtêm informações através do acesso ao arquivo de banco de dados da aplicação armazenado no dispositivo *Android*.;
- Injeção de comando SQL: O atacante insere, modifica ou visualiza informações, de forma não autorizada, no banco de dados do *software* Monitor através da injeção de comandos SQL.;
- Manipulação não autorizada da informação transmitida: O atacante modifica informações, de forma não autorizada, através de ataque de *Man-in-the-middle* na conexão *bluetooth*.;

- Manipulação não autorizada da informação armazenada no banco de dados: O atacante modifica informações, de forma não autorizada, através de acesso direto ao arquivo do banco de dados do *software* Monitor.

Apesar da definição de casos de abusos para o estudo de caso, cabe ressaltar que outras falhas não documentadas previamente podem ser encontradas durante a revisão de código fonte e o teste de penetração. Mais detalhes sobre os casos de abuso definidos para a aplicação podem ser encontrados no Apêndice 1.

4.4 NÍVEL DE CRITICIDADE DAS VULNERABILIDADES ENCONTRADAS

Para identificar o impacto da exploração das vulnerabilidades encontras foram definidos em níveis de criticidade. Estes níveis de criticidade estão relacionados com tamanho do impacto e a facilidade de exploração dos problemas presentes no *software* do estudo de caso.

Esta é uma medida subjetiva, tendo em vista que o impacto e a facilidade de exploração das vulnerabilidades depende do tamanho da organização, seu nicho de negócio, da habilidade do atacante, entre outras características.

A criticidade dos problemas encontrados pode ser dividida em quatro níveis, a seguir:

- **Muito alta:** Causa grande impacto ao negócio da organização com prejuízo financeiro incalculável, grande vazamento de informação confidencial, risco alto de lesão ao operador/paciente assistido e altíssimo comprometimento da imagem da organização;
- **Alta:** Causa grande impacto ao negócio da organização com grave prejuízo financeiro mensurável, vazamento de informação restrita, risco mediano de lesão ao operador/paciente assistido e alto comprometimento da imagem da organização;
- **Media:** Causa impacto ao negócio da organização com baixo prejuízo financeiro, vazamento relevante de informação, comprometimento mediano da imagem da organização, baixo risco de lesão ao operador/paciente assistido;
- **Baixa:** Causa impacto ao negócio da organização com baixo ou nenhum prejuízo financeiro, vazamento irrelevante de informação, baixo comprometimento da imagem da organização, sem risco de lesão ao operador/paciente assistido.

4.5 RELATÓRIO DE ANÁLISE ESTÁTICA DE SEGURANÇA EM SOFTWARE

Após a fase de codificação, foi realizada a revisão de código fonte com o objetivo de verificar se as questões endereçadas pelos requisitos de segurança levantados foram mitigadas no código-fonte.

A verificação realizada levou em consideração os fluxos básicos da aplicação disponibilizados pela sua interface, os requisitos de segurança de *software* levantados e os casos de abuso elicitados.

Mais detalhes sobre os problemas encontrados na revisão de código, evidências e ações recomendadas para a mitigação podem ser encontrados no Apêndice 2. Os problemas de segurança encontrados após a revisão detalhada do código fonte podem ser observados na Tabela 2.

Tabela 2 - Problemas encontrados na revisão de código fonte

#	Vulnerabilidade	Perspectiva de Segurança	Criticidade
01	*Ausência de mecanismo para autenticação de usuários na aplicação*	Autenticação	Muito Alta
02	*Acesso ao banco de dados SQLite sem proteção por senha*	Configuração Segura	Alta
03	*Ausência de mecanismo de armazenamento de informações sensíveis fora do dispositivo Android*	Proteção da Informação	Media
04	*Ausência de mecanismo de backup de informações sensíveis que possa ser utilizado pelo usuário do software Monitor*	Proteção da Informação	Baixa
05	*Ausência de mecanismo de criptografia para as informações sensíveis armazenadas pelo software Monitor no dispositivo Android*	Proteção da Informação	Alta
06	*Ausência de mecanismo para a garantia de autenticidade do sensor na realização da conexão bluetooth*	Comunicação Segura	Alta
07	*Ausência de API de validação de dados inseridos pelo usuário na aplicação*	Validação de Entrada e Saída	Muito Alta
08	*Troca de informação às claras entre o sensor e o dispositivo Android*	Comunicação Segura	Alta
09	*Ausência de mecanismo de responsabilização da atividade realizada no software Monitor*	Responsabilização (*Logging*)	Media

Com uma velocidade de aproximadamente 200 linhas de código por hora, foram verificados aproximadamente 65% do código-fonte em 108 horas de trabalho. Abaixo, na Tabela 3 podem ser visualizadas as métricas básicas do projeto, como a quantidade de classes, quantidade de linhas de código e quantidade de linhas de código em métodos.

Tabela 3 - Métricas básicas do projeto Monitor

Tipo / Objeto	Implementadas Manualmente	Implementadas por Geração Automática	Total
Classes	112	09	121
Linhas de Código	32274	240	32514
Linhas de Código em Métodos	26291	N/A	26291

Cabe ressaltar que foram considerados como geração automática apenas classes e métodos gerados em tempo de compilação. Métodos que utilizam geração automatizada, mas dependem de ação do desenvolvedor (como o caso dos métodos *getters* e *setters*), foram considerados como implementação manual.

4.6 RELATÓRIO DE ANÁLISE DINÂMICA DE SEGURANÇA EM SOFTWARE

Após a revisão de código-fonte, foi realizado um teste de penetração na aplicação. O objetivo era verificar se os itens pontuados na revisão de código fonte e os casos de abuso levantados eram realmente exploráveis.

Foram despendidas aproximadamente 80 horas no teste de penetração realizado na aplicação e foram revisadas todas as funcionalidades disponibilizadas para o usuário. Adicionalmente, foram realizados testes sobre os parâmetros de configuração que a aplicação exige do dispositivo *Android* para seguir segura no ambiente de operação.

Mais detalhes sobre os problemas encontrados no teste de penetração, evidências e ações recomendadas para a mitigação podem ser encontrados no Apêndice 3. Os problemas de segurança encontrados após o teste de penetração podem ser verificados na Tabela 4.

Tabela 4 - Problemas encontrados no teste de penetração

#	Vulnerabilidade	Perspectiva de Segurança	Criticidade
01	*Classes não ofuscadas no arquivo Android Package*	Proteção da Informação	Média
02	*Tamanho inadequado para a senha de conexão bluetooth entre o sensor e o dispositivo Android*	Autenticação	Media
03	*Comunicação bluetooth entre o sensor e o dispositivo Android realizada às claras*	Comunicação Segura	Alta
04	*Arquivo do Banco de Dados SQLite sem criptografia*	Proteção da Informação / Configuração Segura	Muito Alta
05	*Validação insuficiente de dados de entrada*	Validação de Entrada e Saída	Alta

Durante o teste de penetração, os problemas relativos ao caso de abuso de modificação da informação transmitida entre o sensor e o dispositivo *Android* não puderam ser verificados. Isto se deu pela necessidade da utilização de dispositivo especializado para a captura e modificação do trafego *bluetooth* que não estava disponível pela dificuldade de aquisição e pelo custo do equipamento.

Adicionalmente, durante o seu desenvolvimento e implantação o *software* Monitor precisa levar os requisitos e especificações de segurança para a plataforma *Android* listados em Six (2012).

5 DISCUSSÃO E CONCLUSÃO

Este trabalho demonstrou a importância de observar aspectos de segurança no ciclo de desenvolvimento de *software* para dispositivos médicos. De forma geral, os padrões estabelecidos para a regulação e o desenvolvimento de *software* para dispositivos médicos não levam em considerações preocupações com a resiliência do *software* produzido.

É responsabilidade dos QSRs lidar com as preocupações de segurança claramente. Em geral, padrões para normalização de verificação e validação estão somente preocupados com os aspectos funcionais da operação do *software.* Questões de segurança são geralmente problemas colaterais que persistem em todas as fases do ciclo de vida do *software,* até que ele seja retirado de circulação.

Ao estabelecer boas práticas e um fluxo de execução dentro do ciclo de vida do *software* para equipamentos médicos, este trabalho propõe uma metodologia para a construção de aplicações que contempla seguramente os aspectos de segurança de *software* que devem ser mitigados pelas organizações que desenvolvem equipamentos e aplicações médicas.

Com uma abordagem de segurança de *software* abrangente e permanente fica mais simples planejar uma postura adequada de segurança para a aplicação médica. Mitigar os pontos levantados ao longo do ciclo de vida do *software* reduz a chance exploração de problemas de segurança que podem ter consequências catastróficas.

Então, ao avaliar risco, elicitar os seus requisitos de segurança, definir os casos de abuso, realizar a revisão de código fonte e o teste de penetração dentro do contexto do estudo de caso, foi possível enxergar de forma clara possíveis problemas que podem impactar na operação do *software* Monitor. Os pontos encontrados no estudo de caso podem ser explorados facilmente por um *hacker* ou atacante com conhecimento moderado.

Por isso, é possivel concluir que a adoção das boas práticas propostas neste trabalho dentro do contexto da Engenharia Biomédica gera benefícios relacionados a qualidade e a resistência das aplicações médicas contra ataques e explorações de vulnerabilidade de segurança de *software*.

Devido a confidencialidade da informação médica, independente do meio de obtenção (e não somente da informação resultante da relação médico-paciente), os dispositivos médicos devem estabelecer cuidado adicional com a guarda e a transmissão da informação. Deste modo, para toda a informação médica, os aspetos básicos de segurança da informação (confidencialidade, integridade e disponibilidade) não podem ser negligenciados, caracterizando por si só um nicho de preocupação com a segurança da informação e do *software* para equipamentos médicos.

Outro ponto relevante a ser considerado está relacionado com o ambiente de operação do *software*. Organizações que desejam proteger os equipamentos médicos produzidos por elas contra ataques e vazamento de informação, devem estabelecer parâmetros para a operação segura destes equipamentos, oferecendo alternativas para o *hardening* do ambiente de operação e definindo aspectos para a segurança que passam por ajustes de sistema operacional, bancos de dados, serviços externos de gravação da informação, entre outros.

Quando as organizações resolvem desenvolver produtos para plataformas sabidamente problemáticas, elas acatam os riscos de vulnerabilidades que estas plataformas já possuem. Mitigar estes problemas também faz parte do processo de construção de *software* seguro. Então é importante estabelecer e até mesmo corrigir quaisquer problemas que podem levar a explorações no seu equipamento e *software* correlato.

Assim, a adoção desta metodologia minimiza a chance de problemas detectáveis de serem percebidos apenas por agentes mal intencionados e traz esta responsabilidade para a da realidade da organização. Portanto, as suas boas práticas antecipam problemas comuns que podem ser explorados quando a aplicação entra em operação e dá a oportunidade de correção antecipada de vulnerabilidades em momento mais oportuno.

Quando o *software* Monitor começar a seguir um plano de desenvolvimento seguro e se tornar mais maduro, as boas práticas de segurança devem ser aplicadas regularmente. Os padrões utilizados para a regulação do *software* para dispositivos médicos não levam em consideração a preocupação com a segurança. Estes aspectos podem fazer toda a diferença na segurança final do *software* e na proteção contra lesões ao paciente usuário do equipamento médico.

Quando um *software*, como o Monitor, não é planejado e, por consequência, construído com preocupações de segurança, qualquer tipo de problema pode ser encontrado durante a sua avaliação. Como a aplicação avaliada está em um estágio inicial do desenvolvimento, é mais fácil mapear problemas, falhas, questões e vulnerabilidades, bem como criar um plano para mitigação dos artefatos encontrados. O resultado das avaliações realizadas neste trabalho tem condições de apoiar a equipe de desenvolvimento do estudo de caso a alcançar uma maior maturidade no *software* resultante.

Com esta avaliação é possível determinar que a metodologia resultante deste trabalho é adequada para ser utilizada no contexto da engenharia biomédica e da construção dos equipamentos médicos. Porém, não foi possível determinar o custo real para a aplicação desta metodologia nas organizações, pois isto depende do tipo de ciclo de desenvolvimento de *software* adotado, do nível de conhecimento e do tamanho das equipes de requisitos e desenvolvimento, da maturidade dos processos da organização, entre outras características intrínsecas.

Infelizmente, problemas de segurança só podem ser resolvidos quando a equipe inteira envolvida na construção do *software* está consciente de como eles podem afetar a operação da aplicação. Para criar este tipo de consciência muitas ações são importantes. Porém, somente organizações que possuem uma cultura de segurança e pessoal de segurança com habilidades nas boas práticas apresentadas neste trabalho e que implementem um ciclo de desenvolvimento de *software* seguro podem endereçar estas ações corretamente.

Então, este trabalho cumpre os seus objetivos porque a partir da avaliação do estudo de caso foi possível antecipar a revelação de problemas de segurança que podem ser utilizados por atacantes para deturpar o funcionamento do *software* desenvolvido e dá a oportunidade de que estas falhas sejam tratadas adequadamente e no tempo certo, sem problemas relacionados a lesões, danos financeiros ou a imagem do produto.

6 TRABALHOS FUTUROS

Como trabalhos futuros, sugerem-se os seguintes:

1. A utilização da metodologia em outros projetos de *software* para dispositivos médicos, para a coleta e o estabelecimento de métricas de tempo, custo e retorno do investimento;
2. O estabelecimento de um modelo de maturidade de segurança de *software* voltado à construção do *software* médico e seus equipamentos;
3. O estabelecimento de um modelo de levantamento de requisitos de segurança de *software* para a área de engenharia biomédica.

No que tange a primeira sugestão, existe a intenção de realizar outras métricas em *software* diferente da análise neste primeiro momento, porém a quantidade de análises realizadas dependerá da disponibilidade de novas aplicações e de pesquisa ou organizações que desejem submeter seus trabalhos a uma avaliação crítica de segurança.

Para a segunda sugestão, uma avaliação das métricas e da rotina das organizações candidatas é um pré-requisito. Como este aspecto depende de avaliações históricas, apenas organizações que já coletem métricas e que já possuam alguma iniciativa de segurança de *software* podem ser utilizadas para este acompanhamento.

A terceira sugestão se dá pela deficiência dos modelos para o levantamento e o mapeamento de requisitos funcionais e não funcionais de segurança de *software* para a área de engenharia biomédica, que possui diversas características *sui generis* que necessitam de mapeamento específico.

REFERÊNCIAS BIBLIOGRÁFICAS

ABREU, C. B. B., Método para aplicação de modelos de melhoria e avaliação do processo de desenvolvimento de *software* em sistemas críticos de segurança, Dissertação de Mestrado, Universidade de São Paulo – USP, 2008.

ABNT ISO IEC 17799. Tecnologia da informação – Técnicas de segurança – Código de prática para a gestão da segurança da informação, 2ª ed, Rio de Janeiro, 2005.

ANDERSON, R. J. Security Engineering: A Guide to Building Dependable Distributed Systems, 2ª ed, Indianapolis: Wiley Publishing inc, 2008.

BACUDIO, A. G. et. al. An Overview of Penetration Testing, International Journal of Network Security & Its Applications (IJNSA), Vol.3, No.6, Novembro 2011.

BATISTA, C. F. A., Métricas de Segurança de *Software*, Dissertação de Mestrado, Pontificia Universidade Catolica do Rio de Janeiro – PUC-RIO, 2007.

BRASIL. Resolução CFM 1821/2007. Diário Oficial da União, Poder Executivo. Seção I, p. 252. Brasília, DF, 23 nov. 2007.

CHANDRA, P. et al. OpenSAMM – Open Software Assurance Maturity Model, OWASP, Creative Commons, Março, 2009.

CLARK, S. S., FU, K., Recent Results in Computer Security for Medical Devices, International ICST Conference on Wireless Mobile Communication and Healthcare (MobiHealth), Outubro, 2011.

DENNING, T. et al. Patients, Pacemakers, and Implantable Defibrillators: Human Values and Security for Wireless Implantable Medical Devices, ACM Conference on Human Factors in Computing Systems – CHI, 2010

EUA. HITECH Act, 2009. Disponível em: <http://www.hhs.gov/ocr/privacy/hipaa/administrative/enforcementrule/enfifr.pdf> Acesso em: 01 jul. 2011.

FDA, Guidance for the Content of Premarket Submissions for Software Contained in Medical Devices. 1ª ed, New Hampshire, 2005.

FILHO, E. L., Uma abordagem de garantia de segurança de *software* para sistemas críticos embarcados, Dissertação de Mestrado, Instituto Técnologico da Aeronáutica – ITA, 2008

FONTOURA, L. M., Processos de Desenvolvimento de *Software* Confiável Baseados em Padrões de Segurança, Dissertação de Mestrado, Universidade Federal de Santa Maria – UFSM, 2001

FRASER, H.; KWON, Y.; NEUER, M. The future of connected health devices, IBM Institute for Business Value, Março 2011.

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

GOERTZEL, K. M et al. Software Security Assurance: A State of the Art Report (SOAR), Information Assurance Technology Analysis Center (IATAC), Julho 2007.

GOLLAKOTA, S. et al., They Can Hear Your Heartbeats: Non-Invasive Security for Implanted Medical Devices, Proceedings of the ACM SIGCOMM 2011 conference, pp. 2-13, Agosto, 2011.

HALL, K. Developing Medical Device Software to IEC 62304, European Medical Device Technology, v. 1, n. 6, Junho 2010.

HALPERIN, D. et al. Security and Privacy for Implantable Medical Devices, Pervasive Computing – IEEE, Vol. 7 No. 1, pp. 30-39, Janeiro-Março, 2007.

HALPERIN, D. et al. Pacemakers and Implantable Cardiac Defibrillators: Software Radio Attacks and Zero-Power Defenses, Proceedings of the IEEE Symposium on Security and Privacy, 2008, pp. 129-142, doi:10.1109/SP.2008.31.

HANNA, S. et al. Take Two Software Updates and See Me in the Morning: The Case for Software Security Evaluations of Medical Devices, 2nd USENIX Workshop on Health Security and Privacy, San Francisco, CA, Agosto, 2011.

HOGLUND, G., MCGRAW, G. Exploiting Software: How to Break Code. Addison-Wesley, 2004.

HOWARD, M., A Process for Performing Security Code Reviews. IEEE Security & Privacy , Vol. 4, pp. 74-79, 2006

HOWARD, M., LEBLANC, D. E. Writing Secure Code, 2nd ed., Microsoft Press, 2002.

HOWARD, M., LIPNER, S. The Security Development Lifecycle. Microsoft Press, 2006.

IEC 62304, Medical device software – Software life cycle processes, 1ª ed, Geneva, 2006.

ISO 27799, Health informatics – Information security management in health using ISO/IEC 27002, 1ª ed, Geneva, 2008.

ISO 14971, Medical devices – Application of risk management to medical devices, 1ª ed, Geneva, 2007.

ISO/IEC 21827, Systems Security Engineering - Capability Maturity Model, 2ª ed, Geneva, 2008

IVANOFF, G. B., Ambientes e Organizações Virtuais: Cultura de Segurança e Regulação entre o Desenvolvimento de Programas Computacionais e Estruturas e Processos Organizacionais, Dissertação de Mestrado, Universidade de São Paulo – USP, 2006.

KNIGHT, J. C. Dependability of embedded systems, ICSE '02 Proceedings of the 24th International Conference on Software Engineering - ACM, pp 685-686, Nova York, 2002.

LONG, F. et al. The CERT Oracle Secure Coding Standard for Java, 1ª ed, Michigan: Pearson Education Inc, 2012.

MCGRAW, G. Software Security, IEEE Security and Privacy, pp 80-83, Março/Abril 2004.

MCGRAW, G. Software security: building security in, Boston: Addison Wesley Professional, 2006.

MCGRAW, G., CHESS, B., MIGUES, S. BSIMM - Buiding Security In Maturity Model, Cigital, Creative Commons, 2012.

MEAD, N. R., HOUGH, E. D., STEHNEY II, T. R. Security Quality Requirements Engineering (SQUARE) Methodology, Carnegie Mellon University, Pittsburgh, 2005.

MELLON**.,** Secure Software Development Life Cycle, 2007. Disponível em: <https://buildsecurityin.us-cert.gov/bsi/articles/knowledge/sdlc/326-BSI.html>. Acesso em: 01 dezembro 2012.

SEI., CMMI for Development Version 1.3, Technical Report, Carnegie Mellon University, Pittsburgh, 2010.

MICROSOFT, Testes de Penetração, 2008. Disponível em: http://msdn.microsoft.com/pt-br/magazine/cc507646.aspx. Acesso em: 02 dezembro 2012.

MINIATI, R. et al., Medical software management: A failure analysis approach for maintenance and safety plan, Engineering in Medicine and Biology Society (EMBC), 2010 Annual International Conference of the IEEE, pp. 454-457, doi 10.1109/IEMBS.2010.5626129, 2010

NUNES, V. et al. Variability Management of Reliability Models in Software Product Lines: an Expressiveness and Scalability Analysis, SBCARS - Simpósio Brasileiro de Componentes, Arquitetura e Reutilização de Software, 2012.

OWASP, OWASP Top Ten – 2010 The Ten Most Critical Web Application Security Risks, CC:OWASP, 2010.

PAUL, M., Official (ISC)2 Guide to the CSSLP, 1ª ed, Florida: CRC Press, 2011.

RADCLIFFE, J., Hacking Medical Devices for Fun and Insulin: Breaking the Human SCADA System, Black Hat Conference, 2011.

RAKITIN, S. R., Coping with Defective Software in Medical Devices Computer Magazine - IEEE Computer Society, v. 39, 2006, n. 4, pp. 40-45, doi: 10.1109/MC.2006.123.

SBIS., Manual de Certificação para Sistemas de Registro Eletrônico em Saúde (S-RES) Versão 3.0. CFM, 2007. Disponível em: http://portal2.tcu.gov.br/portal/pls/portal/docs/769631.PDF. Acesso em: 01 dezembro 2012.

SIX, J. Application Security for the Android Platform, 1ª ed California: O'Reilly Media, Inc, 2012.

STUTTARD, D.; PINTO, M. The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws. 2ª ed Indianapolis: Wiley Publishing inc, 2011.

TRAVASSOS, G. H.; GUROV, D.; AMARAL, E. A. G. Introdução a Engenharia de *Software* Experimental, RT-ES-590/02, COPPE/UFRJ, Rio de Janeiro, 2002.

VOGEL, D. A. Medical Device Software Verification, Validation, and Compliance, Boston: Artech House, 2011.

WOHLIN, C. et al. Experimentation in Software Engineering: an introduction. Kluwer Academic Publishers, USA, 2000.

APÊNDICES

APÊNDICE 1: CASOS DE ABUSO

Projeto:	Monitor	Versão	1.0
Nome:	Captura do trafego entre o sensor e o dispositivo *Android*		
Autor:	Diogo Rispoli	Data:	31/03/2013
Descrição:	O atacante obtêm as informações trocadas na conexão entre o sensor e o dispositivo *Android* para análise futura e roubo de informação.		
Facilidade de Exploração:	Baixa. O atacante deve possuir a capacidade de capturar tráfego *bluetooth* e analisar o seus diversos protocolos.		

Fluxo Básico

fb01: Um atacante observa o trafego de informações entre o sensor e o dispositivo *Android* através do seu posicionamento entre os dois equipamentos (**fb01-1**); Toda a informação coletada através desta comunicação deve ser armazenada em arquivo de captura para análise futura (**fb01-2**); Desta forma, o atacante pode obter informações sensíveis sobre o paciente para utilizá-las em ataques futuros ao *software* Monitor (**fb01-3**).

Fluxos Alternativos

fa01: O atacante pode obter a captura do trafego através do comprometimento do dispositivo *Android*, sem necessidade de realizar esta escuta no ar com equipamento especial (modifica o fluxo **fb01-1**).

Pontos de Captura

pc01: A captura não é possível porque a comunicação é cifrada;
pc02: Ataque de m*an-in-the-middle* não é possível, pois demanda equipamentos especiais.

Projeto:	Monitor	**Versão:**	1.0
Nome:	Vazamento de informação no *software* Monitor		
Autor:	Diogo Rispoli	**Data:**	31/03/2013
Descrição:	O atacante obtêm informações através do acesso ao arquivo de banco de dados da aplicação armazenado no dispositivo *Android*.		
Facilidade de Exploração:	Moderada. O atacante deve possuir habilidades para desenvolver *malware* e/ou deve conseguir burlar as proteções do sistema operacional *Android* ao acesso de arquivos.		

Fluxo Básico

fb01: Um atacante tem acesso às informações armazenadas pelo *software* Monitor através do comprometimento (via *malware*) do dispositivo *Android* e da captura do arquivo de banco de dados criado pela aplicação (**fb01-1**); O arquivo de banco de dados capturado deve ser armazenado fora do dispositivo *Android* para análise futura (**fb01-2**); Desta forma, o atacante pode obter informações sensíveis sobre o paciente e o aplicativo para utilizá-las em ataques futuros ao *software* Monitor (**fb01-3**).

Fluxos Alternativos

fa01: O atacante pode obter o arquivo do banco de dados através do acesso *root* feito no dispositivo pelo usuário, sem necessidade de realizar comprometimento através de *malware* (modifica o fluxo **fb01-1**).

Pontos de Captura

pc01: Não é possível acessar o arquivo de banco de dados, pois ele está cifrado;
pc02: Não é possível acessar o arquivo de banco de dados, pois ele está protegido;
pc03: Não é possível interpretar a informação, pois ela está cifrada.

Projeto:	Monitor	**Versão:**	1.0
Nome:	Injeção de comando SQL		
Autor:	Diogo Rispoli	**Data:**	31/03/2013
Descrição:	O atacante insere, modifica ou visualiza informações, de forma não autorizada, no banco de dados do *software* Monitor através da injeção de comandos SQL.		
Facilidade de Exploração:	Alta. O atacante deve possuir habilidades para escrever consultas SQL.		

Fluxo Básico

fb01: Um atacante insere, modifica ou visualiza informações no banco de dados através da injeção de comando SQL em campos não validados do *software* Monitor (**fb01-1**); Desta forma, o atacante pode obter ou modificar informações sensíveis sobre o paciente e o aplicativo para utilizá-las em ataques futuros ao *software* Monitor (**fb01-2**).

Fluxos Alternativos

fa01: O atacante pode modificar ou visualizar informações sensíveis através do acesso direto ao arquivo de banco de dados criado no dispositivo *Android* pelo *software* Monitor (modifica o fluxo **fb01-1**).

Pontos de Captura

pc01: Não é possível inserir comandos SQL, pois os campos validam a informação inserida contra injeção de comandos;
pc02: Não é possível acessar o arquivo de banco de dados, pois ele está protegido;
pc03: Não é possível inserir comandos SQL, pois as consultas não são construídas dinamicamente no *software* Monitor.

Pontos de Extensão

pe01: Inclui o caso de abuso "Vazamento de informação no *software* Monitor" (no fluxo **fa01**).

Projeto:	Monitor	**Versão:**	1.0
Nome:	Manipulação não autorizada da informação trasmitida		
Autor:	Diogo Rispoli	**Data:**	31/03/2013
Descrição:	O atacante modifica informações, de forma não autorizada, através de ataque de *Man-in-the-middle* na conexão *bluetooth*.		
Facilidade de Exploração:	Baixa. O atacante deve possuir a capacidade de realizar ataque de *Man-in-the-middle* na conexão *bluetooth*. Este ataque necessita de equipamento especializado e *software* desenvolvido para esta finalidade.		

Fluxo Básico

fb01: Um atacante modifica informações através da manipulação da conexão *bluetooth* entre o sensor e o dispositivo *Android* (**fb01-1**); Desta forma, o atacante pode se colocar entre o sensor e o dispositivo em um ataque conhecido como *Man-in-the-middle* (**fb01-2**).

Fluxos Alternativos

fa01: O atacante pode enviar informações falsas para o *software* Monitor (modifica o fluxo **fb01-1**) através de conexão realizada em um sensor forjado.

Pontos de Captura

pc01: Não é possível modificar as informações, pois a conexão é cifrada;
pc02: Não é possível enviar informações falsas, pois a conexão entre o sensor e o dispositivo é autenticada.
pc03: Não é possível realizar o ataque de *Man-in-the-middle*, pois não existem recursos necessários para este fim.

Pontos de Extensão

pe01: Inclui o caso de abuso "Captura do trafego entre o sensor e o dispositivo *Android*" (no fluxo **fb01**).

Pré-condições

pre01: Necessário equipamento capaz de realizar ataque de *Man-in-the-middle* na conexão *bluetooth.*
pre02: Necessário *software* capaz de modificar informações na conexão *bluetooth* durante o ataque de *Man-in-the-middle* .

Projeto:	Monitor	**Versão:**	1.0
Nome:	Manipulação não autorizada da informação armazenada no banco de dados		
Autor:	Diogo Rispoli	**Data:**	31/03/2013
Descrição:	O atacante modifica informações, de forma não autorizada, através de acesso direto ao arquivo do banco de dados do *software* Monitor.		
Facilidade de Exploração:	Baixa. O atacante deve conseguir burlar as proteções do sistema operacional *Android* ao acesso de arquivos.		

Fluxo Básico

fb01: Um atacante modifica informações através da manipulação do arquivo de banco de dados do *software* Monitor (**fb01-1**); Desta forma, o atacante pode inserir, modificar ou excluir informações armazenadas no dispositivo *Android* (**fb01-2**).

Pontos de Captura

pc01: Não é possível modificar as informações, pois o arquivo é cifrado;
pc02: Não é possível modificar as informações, pois o arquivo é protegido;
pc03: Não é possível interpretar a informação, pois ela está cifrada.

Pontos de Extensão

pe01: Inclui o caso de abuso "Vazamento de informação no *software* Monitor" e "Injeção de comando SQL" (no fluxo **fb01**).

APÊNDICE 2: RELATÓRIO DE REVISÃO DE CÓDIGO-FONTE

Relatório de Análise Estática de Segurança em Software

Projeto:	Monitor	Versão	1.0
Nome:	Relatório de Análise Estática de Segurança em Software		
Avaliador:	Diogo Rispoli	Data:	31/03/2013

Sumário

Introdução

A análise estática de segurança, também chamada de revisão de código-fonte, é um instrumento importante para verificar se as questões de segurança endereçadas pelos requisitos foram mitigadas durante a fase de implementação do projeto. Esta verificação é realizada por meio de inspeção linha-a-linha do código fonte da aplicação.

A revisão realizada no *software* Monitor levou em consideração as seguintes perspectivas de segurança de *software*: Autenticação, gerenciamento de sessão, validação de entrada e saída, comunicação segura, proteção da informação, responsabilização (*logging*) e configuração segura.

Detalhes relativos a detalhes da revisão realizada estão presentes no Anexo I – Detalhes da Revisão de Código-Fonte. Neste anexo, é possível encontrar uma descrição do problema encontrado, sugestões de correção ou mitigação e evidências para os problemas apontados.

Os requisitos de segurança de *software* considerados nesta avaliação estática estão presentes no Anexo II – Requisitos de Segurança de *Software*.

Os casos de abuso avaliados durante a revisão de código fonte podem ser encontrados no Anexo III – Casos de Abuso.

Resultados

O resumo dos resultados obtidos nesta avaliação de *software* está presente na Tabela 1.

Tabela 1 – Vulnerabilidades encontradas na avaliação estática de *software*

#	Vulnerabilidade	Perspectiva de Segurança	Criticidade
01	*Ausência de mecanismo para autenticação de usuários na aplicação*	Autenticação	**Muito Alta**
02	*Acesso ao banco de dados SQLite sem proteção por senha*	Configuração Segura	**Alta**
03	*Ausência de mecanismo de armazenamento de informações sensíveis fora do dispositivo Android*	Proteção da Informação	**Media**
04	*Ausência de mecanismo de backup de informações sensíveis que possa ser utilizado pelo usuário do software Monitor*	Proteção da Informação	**Baixa**
05	*Ausência de mecanismo de criptografia para as informações sensíveis armazenadas pelo software Monitor no dispositivo Android*	Proteção da Informação	**Alta**
06	*Ausência de mecanismo para a garantia de autenticidade do sensor na realização da conexão bluetooth*	Comunicação Segura	**Alta**
07	*Ausência de API de validação de dados inseridos pelo usuário na aplicação*	Validação de Entrada e Saída	**Muito Alta**
08	*Troca de informação às claras entre o sensor e o dispositivo Android*	Comunicação Segura	**Alta**
09	*Ausência de mecanismo de responsabilização da atividade realizada no software Monitor*	Responsabilização (*Logging*)	**Media**

Diante dos problemas encontrados nesta avaliação, recomenda-se:

- Correção urgente de todos os problemas com criticidade Alta e Muito Alta;
- Treinamento da equipe de requisitos em técnicas de elicitação de requisitos de *software* seguro;
- Treinamento da equipe de desenvolvedores em técnicas de construção de *software* seguro.

Conclusão

A análise realizada encontrou problemas que devem ser remediados no código-fonte da aplicação. Alguns dos problemas encontrados estão relacionados a falhas no processo de levantamento de requisitos de segurança da aplicação.

Desta forma, recomenda-se a correção de todos os itens levantados, prioritariamente os marcados com criticidade alta e muito alta. Recomenda-se também uma reavaliação dos requisitos de segurança da aplicação, levando-se em consideração os requisitos de segurança apontados no Anexo II – Requisitos de Segurança de Software.

Anexo I – Detalhes da Revisão de Código-Fonte

Criticidade dos Problemas Encontrados

A criticidade dos problemas encontrados pode ser dividida em quatro níveis, a seguir:

- **Muito alta** – Causa grande impacto ao negócio da organização com prejuízo financeiro incalculável, grande vazamento de informação confidencial, risco alto de lesão ao operador/paciente assistido e altíssimo comprometimento da imagem da organização;
- **Alta** – Causa grande impacto ao negócio da organização com grave prejuízo financeiro mensurável, vazamento de informação restrita, risco mediano de lesão ao operador/paciente assistido e alto comprometimento da imagem da organização;
- **Media** – Causa impacto ao negócio da organização com baixo prejuízo financeiro, vazamento relevante de informação, comprometimento mediano da imagem da organização, baixo risco de lesão ao operador/paciente assistido;
- **Baixa** - Causa impacto ao negócio da organização com baixo ou nenhum prejuízo financeiro, vazamento irrelevante de informação, baixo comprometimento da imagem da organização, sem risco de lesão ao operador/paciente assistido.

Métricas

Total de classes do projeto Monitor : 121 classes
Classes implementadas manualmente*: 112 classes
Classes implementadas por geração automática*: 9 classes

Total de linhas de código: 32514 linhas
Linhas implementadas manualmente*: 32274 linhas
Linhas implementadas por geração automática*: 240 linhas

Total de linhas de código em métodos: 26291 linhas
Linhas implementadas manualmente*: 26291 linhas
Linhas implementadas por geração automática*: 0 linhas

Número total de métodos do projeto Monitor: 1691
Métodos implementados manualmente*: 1691 métodos
Métodos implementados por geração automática*: 0 métodos

*Foi considerado como geração automática apenas classes e métodos gerados em tempo de compilação. Métodos que utilizam geração automatizada, mas dependem de ação do desenvolvedor (como o caso dos métodos *getters* e *setters*), foram considerados como implementação manual.

Ausência de mecanismo para autenticação de usuários na aplicação

Descrição

O *software* Monitor não possui mecanismo de arquitetura para autenticação de usuário. Por este motivo, qualquer usuário que tenha acesso ao aparelho pode modificar as configurações do *software* Monitor ou visualizar as informações disponibilizadas aos usuários, sejam elas sensíveis ou não.

Ações Recomendadas

Desenvolver mecanismo de autenticação que solicite a autenticação do usuário antes de acessar o *menu* da aplicação;

Armazenar em banco de dados as credenciais de acesso privadas (senhas) na forma de *hash* criptográfico, utilizando algoritmo SHA-2 ou superior, acrescida de *salt* para evitar ataques de dicionário ou *Rainbow Tables*;

Preferencialmente armazenar as credenciais de acesso em banco de dados externo ao dispositivo *Android*. Desta forma, o acesso à informação sensível é minimizado. Caso as credenciais de acesso fiquem armazenadas fora do equipamento, o processo de autenticação deve utilizar conexão segura (SSL) com processamento das informações no servidor fora do dispositivo *Android*.

Evidências

N/A.

Acesso ao banco de dados SQLite sem proteção por senha

Descrição

O acesso ao arquivo de banco de dados criado pela aplicação não utiliza proteção por senha.

O Banco de dados SQLite é um banco de dados leve utilizado em aplicações escritas em Android. Este banco de dados pode ser armazenado no próprio dispositivo *Android* e provê um mecanismo rápido e confiável para o acesso a dados que necessitam ser persistidos.

Ações Recomendadas

Utilização de API de criptografia SQLCipher para criptografar o arquivo do banco de dados e adicionar proteção de senha ao acesso.

Utilizar ofuscador de código para evitar roubo de informações sensíveis de acesso ao banco de dados criado pela aplicação no dispositivo *Android*.

Evidências

Classe: /Monitor/src/br/unb/mestrado/monitor/persistence/MechanismSQLite.java

Ausência de mecanismo de armazenamento de informações sensíveis fora do dispositivo Android

Descrição

As informações coletadas pelo sensor e armazenadas no dispositivo *Android* não são transferidas para servidor centralizado que armazene estas informações de forma completa e segura, realizando um histórico real das informações coletadas.

Ações recomendadas

As informações armazenadas pelo *software* Monitor no dispositivo *Android* devem permanecer por um período temporário no dispositivo, a ser definido de acordo com as necessidades de comparação histórica da saúde do paciente e da permanência sem conexão com o servidor centralizado da aplicação.

Evidências

N/A.

Ausência de mecanismo de backup de informações sensíveis que possa ser utilizado pelo usuário do software Monitor

Descrição

O usuário do *software* Monitor não pode realizar backup das informações coletadas pelo sensor fora do dispositivo *Android*.

Ações Recomendadas

Deve ser disponibilizada interface com *software* que possa ser utilizado pelo paciente para realizar *backup* privado das informações coletadas pelo *software* Monitor. Este mecanismo de *backup* deve possuir características de segurança que permitam que a informação extraída não fique disponível a usuários maliciosos sem proteção.

O software de *backup* deve armazenar as informações do paciente de forma criptografada, utilizando algoritmos de criptografia modernos e sem vulnerabilidades conhecidas, para que estas informações não possam ser acessadas indiscriminadamente.

Evidências

N/A.

Ausência de mecanismo de criptografia para as informações sensíveis armazenadas pelo software Monitor no dispositivo Android

Descrição

Se um atacante comprometer as proteções de segurança disponibilizadas pelo sistema operacional *Android*, seja com *malware* ou elevando o seu privilégio de execução, este atacante pode ter acesso livre a qualquer informação armazenada no dispositivo. Por este motivo, informações sensíveis armazenadas às claras podem ser ficar disponíveis ao atacante.

Ações Recomendadas

Integração de mecanismo de criptografia à base de dados SQLIte como SQLCipher ou equivalente.

SQLCipher é um projeto *Open Source* para a criptografia de banco de dados leve SQLite, suportado em diversas plataformas, entre elas o *Android*. SQLCipher provê uma forma organizada de adicionar mecanismos de criptografia a base de dados criadas por aplicativos e que ficam armazenadas no dispositivo *Android*.

Evidências

Classe: /Monitor/src/br/unb/mestrado/monitor/persistence/MechanismSQLite.java

Ausência de mecanismo para a garantia de autenticidade do sensor na realização da conexão bluetooth

Descrição

Ao realizar a conexão *bluetooth* com o sensor, o *software* Monitor não consegue reconhecer se esta conexão é realizada com sensor autêntico ou forjado. A utilização da senha para estabelecer esta conexão não garante que o usuário está se conectando com dispositivo correto.

Ações Recomendadas

Ao realizar a conexão com o sensor *bluetooth*, deve ser realizada garantia da autenticidade baseada na utilização de certificado digital armazenado no sensor.

Evidências

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    this.startBluetoothChannelService();
}

protected void onDestroy() {

private void startBluetoothChannelService() {
    Intent bluetoothServiceIntent = new Intent(ActivityIntent.START_BLUETOOTH_SERVICE);
    bluetoothServiceIntent.setClass(getApplicationContext(), BluetoothManagerService.class);
    this.startService(bluetoothServiceIntent);
    this.bindService(bluetoothServiceIntent, connection, BIND_AUTO_CREATE);
}
```

Figura 1 – Trecho de código onde é realizada solicitada conexão com o sensor *bluetooth*

Classe:
/Monitor/src/br/unb/mestrado/monitor/activity/bluetooth/BluetoothSettingActivityInterface.java

```
public void onStart(Intent intent, int startId) {
    super.onStart(intent, startId);
    this.registerBluetoothListener();
    this.registerServiceListener();
    this.startProtocolService();
}

public void onDestroy() {

 * Inicia uma comunicação com o serviço de protocolo.
private void startProtocolService() {
    Intent intentService = new Intent(this,BluetoothParserService.class);
    this.startService(intentService);
}

private void registerBluetoothListener() {
  IntentFilter filterBluetoothAPI = new IntentFilter();
  filterBluetoothAPI.addAction(BluetoothDevice.ACTION_FOUND);
  filterBluetoothAPI.addAction(BluetoothDevice.ACTION_ACL_DISCONNECTED);
  filterBluetoothAPI.addAction(BluetoothAdapter.ACTION_DISCOVERY_FINISHED);
  filterBluetoothAPI.addAction(BluetoothAdapter.ACTION_DISCOVERY_STARTED);
  this.registerReceiver(bluetoothBroadcastReceiver, filterBluetoothAPI);
}

private void registerServiceListener(){
    IntentFilter serviceListenerAPI = new IntentFilter();
    serviceListenerAPI.addAction(ActivityIntent.STOP_ALL_SERVICES);
    this.registerReceiver(serviceBroadcastReceiver,serviceListenerAPI);
}
```

Figura 2 – Trecho de código onde é realizada a conexão com o sensor *bluetooth*

Classe:
/Monitor/src/br/unb/mestrado/monitor/cm/communication/connection/impl/BluetoothManagerService.java

Ausência de API de validação de dados inseridos pelo usuário na aplicação

Descrição

A aplicação não possui estratégia de verificação de dados inseridos na aplicação. Desta forma, as informações inseridas pelo usuário na aplicação não são validadas contra ataques de injeção de comando.

Ações recomendadas

Criação de componente arquitetural para validação de entrada de dados contra ataques de injeção de comando. Este componente deve utilização técnicas de validação de entrada lista negra / lista branca, dependendo do tipo de informação a ser validada na aplicação, de forma centralizada.

Evidências

```
        //BTN SALVAR
        ((Button)findViewById(R.id.btn_save)).setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View view) {
                try {
                    mountUser();
                    UserControl control = BusinessObjectFactory.getUserControl();
                    control.insert(user);
                    AccountMaintainActivity.this.setResult(USER_INSERTED);
                    AccountMaintainActivity.this.finish();
                } catch (BusinessException e) {
                    showErroMessage(e);
                }
            }
        });

        ((Button)findViewById(R.id.btn_cancel)).setOnClickListener(new OnClickListener() {
    }

    protected void configureControl() {

    protected void mountUser() {
        Spinner spinner = (Spinner) findViewById(R.id.cmb_account_type);
        if(spinner.getSelectedItem()!=null){
            AccountType type = AccountType.values()[spinner.getSelectedItemPosition()];
            this.user.setType(type);
        }
        this.user.setNotify(((CheckBox)findViewById(R.id.chk_notify)).isChecked());
    }
```

Figura 3 – Trecho de código onde é solicitada a inserção de usuário na base de dados do *software* Monitor

Classe: /Monitor/src/br/unb/mestrado/monitor/activity/register/AccountMaintainActivity.java

```java
public void insert(User user) throws BusinessException {
    this.validate(user);
    try {
        if (user.getId() != null && user.getId() > 0) {
            this.update(user);
        } else {
            this.userDAO.insert(user);
        }
    } catch (PersistenceException e) {
        throw new BusinessException("Erro de persistência",
                "Não foi possivel inserir usuário.", e);
    }
}

private void validate(User user) throws BusinessException {
    StringBuilder sb = new StringBuilder("");
    if (user.getName() == null || user.getName().equals("")) {
        sb.append("Nome do usuário é obrigatório ou não é válido.\n");
    }

    if (user.getType() == null) {
        sb.append("Tipo do usuário é obrigatório ou não é válido.\n");
    }
    if (!sb.toString().equals("")) {
        throw new BusinessException("Erro de validação", sb.toString());
    }
}
```

Figura 4 – Classe de negócio onde é realizada a inserção no bando de dados

Classe: /Monitor/src/br/unb/mestrado/monitor/control/business/UserControl.java

A classe /Monitor/src/br/unb/mestrado/monitor/common/model/bdmanipulation/User.java contem somente métodos *getters* and *setters* que não possuem validação.

```java
public class User extends Entity{

    private static final long serialVersionUID = 1L;
    private String name;
    private Long idMobile;
    private boolean notify;
    private AccountType type;

    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public Long getIdMobile() {
    public void setIdMobile(Long idMobile) {
    public boolean getNotify() {
    public void setNotify(boolean notify) {
    public void setType(AccountType type) {
    public AccountType getType() {
}
```

Figura 5 – Classe User.Java

Classe: /Monitor/src/br/unb/mestrado/monitor/common/model/bdmanipulation/User.java

Troca de informação às claras entre o sensor e o dispositivo Android

Descrição

As informações trocadas entre o sensor e o dispositivo *Android* são enviadas as claras. Desta forma, um atacante pode interceptar a comunicação e modificar as informações em um ataque de *Man-in-the-middle*.

Apesar das informações coletadas serem trocadas em formato próprio utilizado pelo *software* Monitor, o que dificultaria o entendimento do padrão da informação quando coletada no ar, pela decompilação das classes do *software* Monitor esta informação pode ficar acessível em formato que pode ser entendido pelo atacante.

Ações Recomendadas

Utilização do sistema de criptografia utilizando pelo *bluetooth*, em todos os equipamentos que tem poder computacional e suporte para este fim. Adicionalmente e/ou alternativamente, pode ser adotado um padrão para a criptografia da mensagem trocada implementado pelo *software* Monitor.

Evidências

Classes:
/Monitor/src/br/unb/mestrado/monitor/cm/communication/connection/impl/BluetoothManagerService.java

/Monitor/src/br/unb/mestrado/monitor/activity/bluetooth/BluetoothSettingActivityInterface.java

Ausência de mecanismo de responsabilização da atividade realizada no software Monitor

Descrição

A aplicação não possui mecanismo que armazena informações sobre a atividade realizada no *software* Monitor. Mecanismos de responsabilização são utilizados para determinar a atividade do usuário no *software*, como por exemplo, data e horário do inicio e fim da monitoração, informações sobre o sensor utilizado, como o seu BD_ADDR, informações sobre data e horário de ajustes e modificações nas configurações da aplicação e do sensor, nome do usuário, histórico de alertas, histórico de comunicação com o servidor central do software, entre outros. Estes registros são úteis para a resolução de problemas de *software*, erros, identificação de ataques, registro de falhas e interrupção de comunicação no serviço, utilização não autorizada, não-repudio, entre outros.

Ações Recomendadas

Criação de mecanismo de responsabilização que estejam de acordo com as normas IEC 62304 (*Medical device software - Software life cycle processes, 2006*) e ISO 27799 (*Health informatics – Information security management in health using ISO/IEC 27002, 2008*) e que possa ser sincronizado com servidor centralizado da aplicação. Cabe ressaltar que as informações de responsabilização precisam ser armazenadas de forma segura para evitar interferências de agentes externos.

Evidências

N/A.

Anexo II – Requisitos de Segurança de Software

Os seguintes requisitos de segurança de *software* foram levados em consideração durante o processo de revisão de código-fonte:

- Autenticação segura;
- Transmissão de informação criptografada entre o sensor e o dispositivo *Android*;
- Proteção do arquivo da base de dados local;
- Validação de entrada contra injeção de comando;
- Alteração do código de conexão com o sensor;
- Armazenamento seguro de dados sensíveis;
- Armazenamento seguro de credenciais;
- Registro das atividades do usuário no *software* Monitor.

Anexo III – Casos de Abuso

Os casos de abuso se baseiam nos requisitos de segurança porque eles são um guia de como verificar se os requisitos de segurança foram implementados. Desta forma, os casos de abuso que foram definidos para o software Monitor, com uma breve descrição da exploração que será avaliada, são:

- Captura do tráfego entre o sensor e o dispositivo *Android*: O atacante obtêm as informações trocadas na conexão entre o sensor e o dispositivo *Android* para análise futura e roubo de informação;
- Vazamento de informação no *software* Monitor: O atacante obtêm informações através do acesso ao arquivo de banco de dados da aplicação armazenado no dispositivo *Android*.;
- Injeção de comando SQL: O atacante insere, modifica ou visualiza informações, de forma não autorizada, no banco de dados do *software* Monitor através da injeção de comandos SQL.;
- Manipulação não autorizada da informação transmitida: O atacante modifica informações, de forma não autorizada, através de ataque de *Man-in-the-middle* na conexão *bluetooth*.;
- Manipulação não autorizada da informação armazenada no banco de dados: O atacante modifica informações, de forma não autorizada, através de acesso direto ao arquivo do banco de dados do *software* Monitor.

APÊNDICE 3: RELATÓRIO DE TESTE DE PENETRAÇÃO

Relatório de Análise Dinâmica em Segurança de Software

Projeto:	Monitor	Versão	1.0
Nome:	Relatório de Análise Dinâmica em Segurança de Software		
Testador:	Diogo Rispoli	Data:	15/04/2013

Sumário

Introdução

O teste de penetração é um instrumento importante para verificar a resiliência de um *software* a ataques promovidos por pessoas mal intencionadas. Este teste é realizado com a aplicação sendo executada em ambiente com características análogas ao seu ambiente de produção.

O teste realizado no *software* Monitor levou em consideração as seguintes perspectivas de segurança de *software*: Autenticação, validação de entrada e saída, comunicação segura, proteção da informação / criptografia e configuração segura.

Detalhes relativos a detalhes do teste executado estão presentes no Anexo I – Detalhes dos Testes Realizados. Neste anexo, também são encontradas as evidências ligadas ao resultado dos testes realizados e as possibilidades de exploração das vulnerabilidades confirmadas. O referido anexo também aborda sugestões para correção dos problemas evidenciados.

Os requisitos de segurança de *software* considerados nesta avaliação dinâmica estão presentes no Anexo II – Requisitos de Segurança de *Software*.

Os casos de abuso avaliados durante o teste de penetração podem ser encontrados no Anexo III – Casos de Abuso.

Resumo Executivo

O resumo dos resultados obtidos nesta avaliação de *software* está presente na Tabela 1.

Tabela 1 – Vulnerabilidades encontradas na avaliação dinâmica de software

#	Vulnerabilidade	Perspectiva de Segurança	Criticidade
01	*Classes não ofuscadas no arquivo Android Package*	Proteção da Informação	Média
02	*Tamanho inadequado para a senha de conexão bluetooth entre o sensor e o dispositivo Android*	Autenticação	Media
03	*Comunicação bluetooth entre o sensor e o dispositivo Android realizada às claras*	Comunicação Segura	Alta
04	*Arquivo do Banco de Dados SQLite sem criptografia*	Proteção da Informação / Configuração Segura	Muito Alta
05	*Validação insuficiente de dados de entrada*	Validação de Entrada e Saída	Alta

Diante dos problemas encontrados nesta avaliação, recomenda-se:

- Correção urgente de todos os problemas com criticidade Alta e Muito Alta;
- Treinamento da equipe de requisitos em técnicas de elicitação de requisitos de *software* seguro;
- Treinamento da equipe de desenvolvedores em técnicas de construção de *software* seguro.

Conclusão

A análise realizada encontrou problemas que devem ser remediados no processo de desenvolvimento do *software* Monitor. Os problemas encontrados podem levar ao vazamento de informação sensível e a ataques aos serviços correlatos do *software* Monitor. Alguns dos problemas encontrados estão relacionados a falhas no processo de levantamento de requisitos e na avaliação de riscos de segurança da aplicação.

Desta forma, recomenda-se a correção de todos os itens levantados, prioritariamente os marcados com criticidade alta e muito alta. Recomenda-se também uma reavaliação dos requisitos de segurança da aplicação, levando-se em consideração os requisitos de segurança apontados no Anexo II – Requisitos de Segurança de Software.

Anexo I – Detalhes dos Testes Realizados

A criticidade dos problemas encontrados pode ser dividida em quatro níveis, a seguir:

- **Muito Alta** – Causa grande impacto ao negócio da organização com prejuízo financeiro incalculável, grande vazamento de informação confidencial, risco alto de lesão ao operador/paciente assistido e altíssimo comprometimento da imagem da organização;
- **Alta** – Causa grande impacto ao negócio da organização com grave prejuízo financeiro mensurável, vazamento de informação restrita, risco mediano de lesão ao operador/paciente assistido e alto comprometimento da imagem da organização;
- **Media** – Causa impacto ao negócio da organização com baixo prejuízo financeiro, vazamento relevante de informação, comprometimento mediano da imagem da organização, baixo risco de lesão ao operador/paciente assistido;
- **Baixa** - Causa impacto ao negócio da organização com baixo ou nenhum prejuízo financeiro, vazamento irrelevante de informação, baixo comprometimento da imagem da organização, sem risco de lesão ao operador/paciente assistido.

Classes não ofuscadas no arquivo Android Package

Descrição

Não é utilizado nenhum ofuscador de classes na compilação do arquivo *Android Package (.apk)*. Por este motivo, o *software* Monitor está suscetível a decompilação dos seu pacote e possíveis modificações não autorizadas.

Ações Recomendadas

Utilização do *framework* Proguard para a ofuscação dos arquivos antes da compilação do pacote distribuível.

Evidências

Com a utilização do *software* dex2jar, foi extraído o pacote .jar a partir do arquivo .dex presente no pacote .apk do *software* monitor.

A partir disso, foi utilizada a ferramenta JD (*Java Decompiler*) para decompilar os arquivos .class presentes no pacote .jar extraído. Uma comparação ilustrativa entre a classe original e a classe decompilada pode ser realizada pela visualização da Figura 1 e da Figura 2.

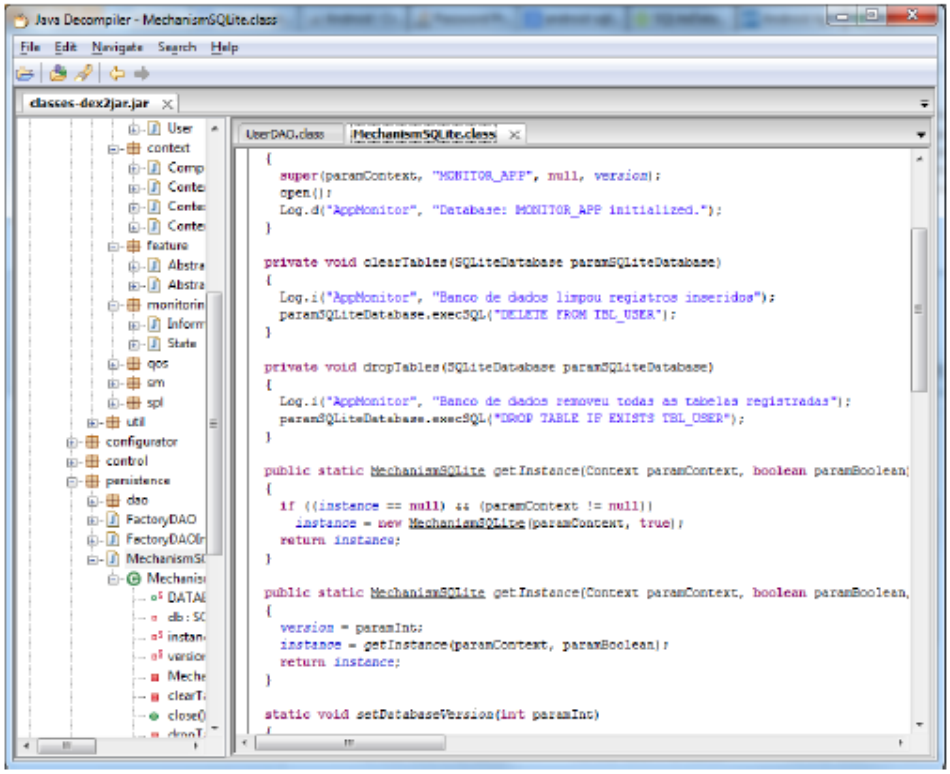

Figura 1 – Classe MechanismSQLite.class decompilada

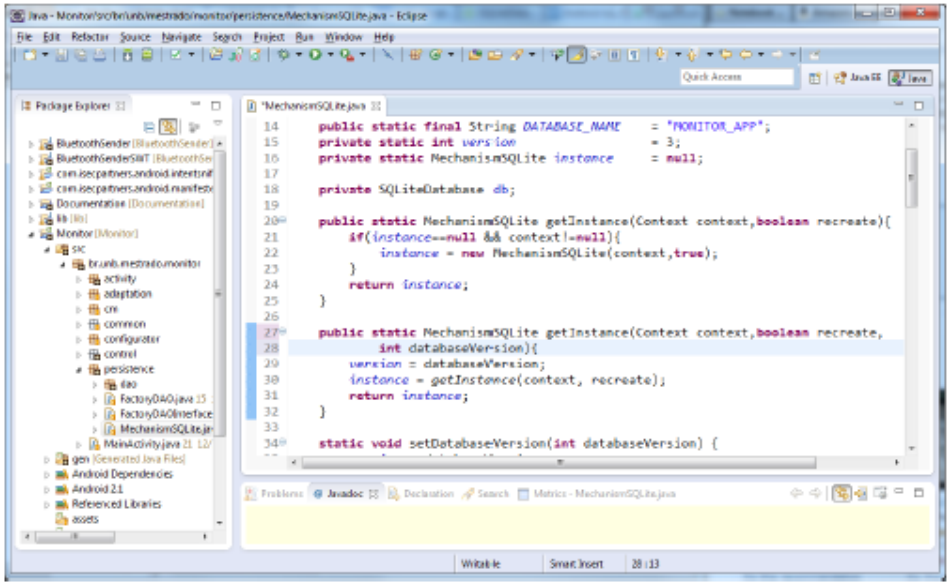

Figura 2 – Classe MechanismSQLite.java original

Tamanho inadequado para a senha de conexão bluetooth entre o sensor e o dispositivo Android

Descrição

A senha utilizada pelo sensor é de apenas 4 (quatro) caracteres. Por este motivo, ela pode ser facilmente computada ou derivada. Podem ser utilizados ataques de força bruta para detectar a senha correta utilizada na comunicação entre o sensor e o dispositivo *bluetooth* e viabilizar ataques de *Man-in-the-middle*.

Ações Recomendadas

Adoção de senha com 8 (oito) caracteres. Criação de mecanismo de troca de senha padrão por senha personalizada pelo usuário do *software* Monitor. Definição de tempo máximo de troca de senha de 3 (três) meses.

Evidências

N/A.

Comunicação bluetooth entre o sensor e o dispositivo Android realizada às claras

Descrição

A comunicação realizada via bluetooth entre o sensor e o dispositivo Android envia os dados às claras. Desta forma, esta comunicação pode ser manipulada e comprometida em um ataque de *Man-in-the-middle*, bem como pode ser utilizada para construir aplicações que atuem como um sensor falso.

Ações Recomendadas

Utilização dos recursos de criptografia da tecnologia *bluetooth* ou implementação via *software* da embaralhamento/criptografia da comunicação entre o sensor e o dispositivo Android.

Evidências

A captura da comunicação bluetooth pode ser realizada de no ar ou posicionado em um dos dispositivos da comunicação. A captura realizada no ar demanda equipamento especializado de comunicação via rádio ou *dongle* USB *bluetooth* customizado. Já a captura realizada no dispositivo de comunicação, depende de *software* especifico. Para a coleta desta evidência foi utilizado o *software* de captura rSAP, disponível para a plataforma Android.

Após a captura, o arquivo com as informações do tráfego entre o sensor e o dispositivo Android foi movido para um computador. Para o análise dos pacotes foi utilizado o analisador de protocolos *Wireshark*. Nesta captura, estava disponível todo o *handshake* da comunicação *bluetooth* e a troca de informação por 5 (cinco) minutos entre o sensor e o dispositivo *bluetooth*, desde a descoberta do sensor até o momento de desconexão de sua desconexão.

Durante a análise do tráfego foi possível observar diversos dados, mas, não foi possível diferenciar dentro dos pacotes RFCOMM (que é o protocolo de transporte da tecnologia *bluetooth*) a que estes dados estavam ligados conforme é possível visualizar na Figura 3. Mesmo assim, foi possível inferir que o sensor encaminha as informação entre 9 (nove) e 11 (onze) posições por vez.

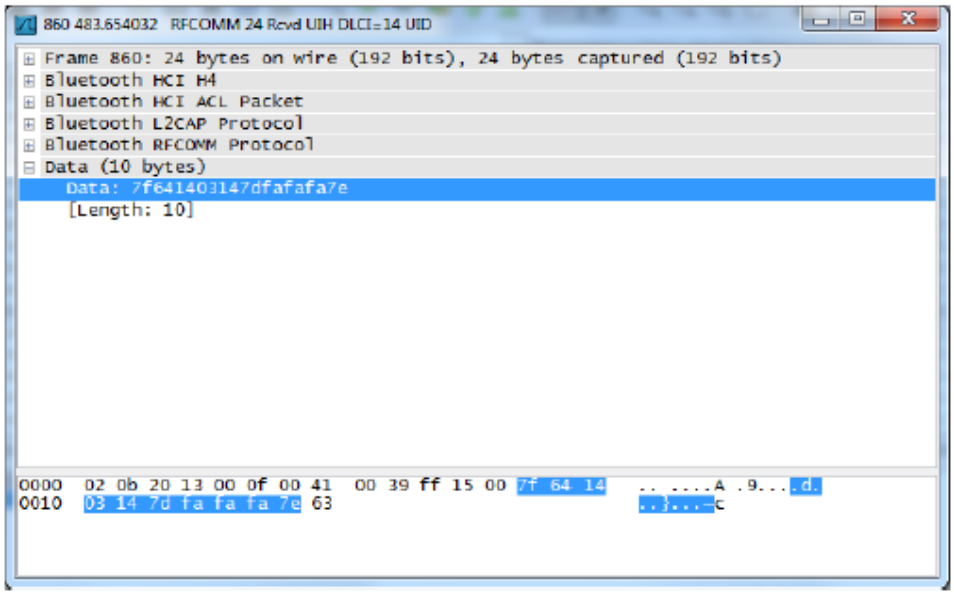

Figura 3 – Exemplo de informação às claras no pacote RFCOMM

O atacante necessita conhecer o padrão de comunicação da aplicação, fato este que pode ser contornado a partir da decompilação do pacote *.apk* do *software* Monitor. Durante a fase da decompilação do pacote Android e da inspeção de suas classes, foi possível obter informações da posição dos dados nos pacotes de transporte do *bluetooth* na interface `br.unb.mestrado.monitor.cm.communication.constant.DataPackageConstant.class`.

Assim, é possível começar determinar o tipo da informação e sua posição dentro do pacote enviado pelo sensor ao dispositivo *Android*. A interface citada não oferece o modo completo de como é consumida pelo *software* Monitor, mas indica a posição inicial e final dos dados e a qual sensor aquela informação pertence (SPO2, ECG ou TEMPERATURA) como se pode visualizar na Figura 4.

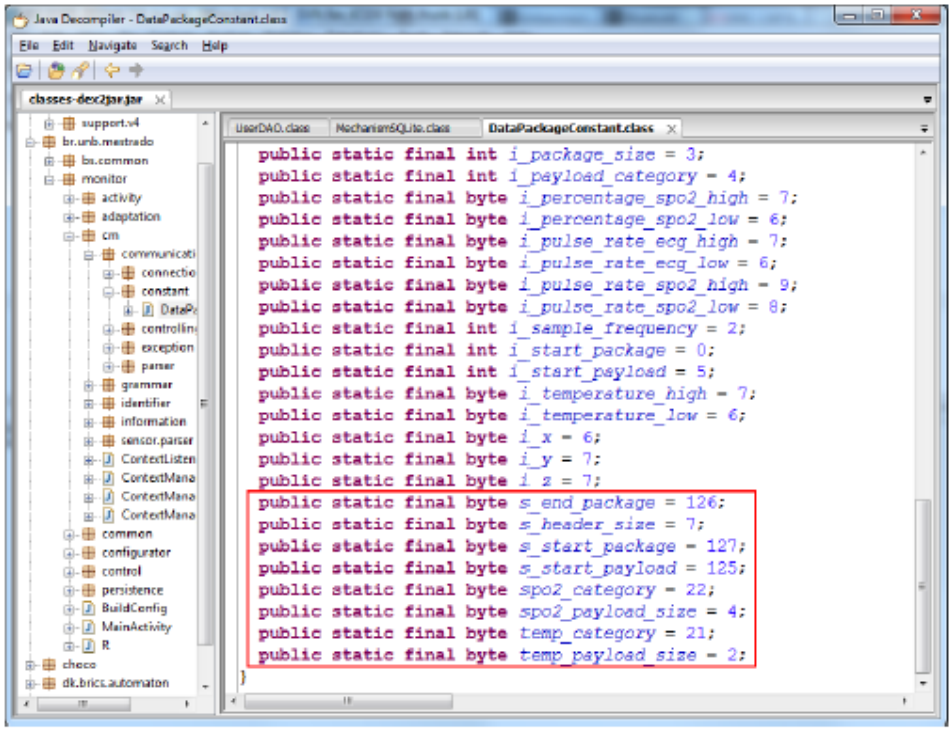

Figura 4 – Posição da informação transmitida pelo sensor ao *software* Monitor no pacote RFCOMM

Os dados visualizados no Wireshark estão em hexadecimal, o que pode demandar o uso de uma calculadora de hexadecimal ou uma tabela ASCII, disponível no sistema operacional ou pela *internet*.

Para compreender a posição dos dados e como o *software* Monitor realiza o *parser* destas informações, podem ser consultadas as classes decompiladadas do pacote `br.unb.mestrado.monitor.cm.sensor.parser.*`. Um exemplo destas classes pode ser observado na Figura 5.

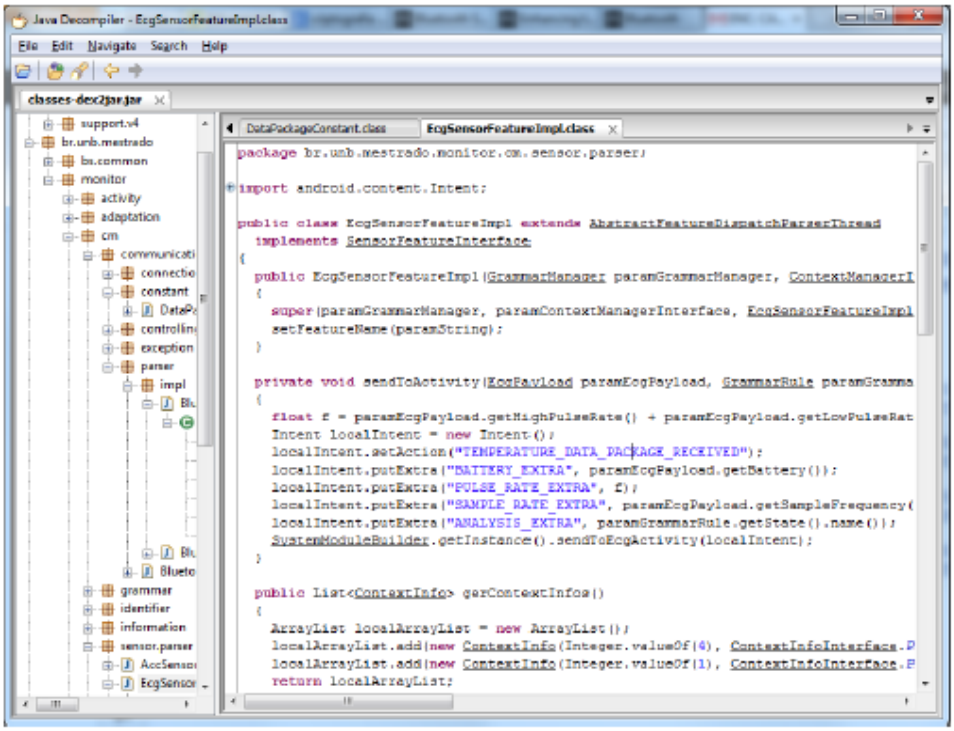

Figura 5 – Classe decompilada ECGSensorFeatureImpl.class

Arquivo do Banco de Dados SQLite sem proteção

Descrição

O arquivo de banco de dados SQL criado pelo *software* Monitor no dispositivo Android não possui mecanismo de proteção dos dados, como a criptografia.

Ações Recomendadas

Utilização do *framework* SQLCipher para proteção criptografia do arquivo de banco de dados SQLite ou utilização de criptografia/ofuscação de informações sensíveis armazenadas no banco de dados.

Evidências

Existem três formas de obter o arquivo de banco de dados armazenado no dispositivo *Android*.

A primeira passa pela utilização do comando `adb pull`, disponível na plataforma Android SDK (*Software Development Kit*) a partir da versão 4 (quatro), que cópia o arquivo de banco de dados para um computador no qual o dispositivo android esteja conectado. É necessário que as opções de desenvolvedor do sistema operacional *Android* estejam habilitadas para a utilização deste comando.

A segunda forma é realizar a cópia programaticamente para o cartão SD (*Secure Digital*) do dispositivo, a partir da manipulação das permissões do sistema *Android* e utilizando o código de exemplo presente na Figura 6. Adicionalmente, pode ser realizado procedimento de engenharia reversa que modifique o *software* Monitor para que ele mesmo realize a cópia da base de dados, sem necessidade de alteração das permissões. Este procedimento pode ser automatizado por meio de *malware*.

```
public class ExtractDBFromAndroid {

    public void extractDB() {
        try {
            File sd = Environment.getExternalStorageDirectory();
            File data = Environment.getDataDirectory();

            if (sd.canWrite()) {
                String currentDBPath = "//data//yourpackagename//databases//yourdatabasename";
                String backupDBPath = "backupname.db";
                File currentDB = new File(data, currentDBPath);
                File backupDB = new File(sd, backupDBPath);

                if (currentDB.exists()) {
                    FileChannel src = new FileInputStream(currentDB).getChannel();
                    FileChannel dst = new FileOutputStream(backupDB).getChannel();
                    dst.transferFrom(src, 0, src.size());
                    src.close();
                    dst.close();
                }
            }
        } catch (Exception e) {

        }
    }
```

Figura 6 – Código de exemplo para a cópia de arquivo no dispositivo *Android*.

A terceira forma é obter acesso de *super* usuário (*root*) a partir da modificação do *firmware* principal do sistema operacional *Android* instalado no dispositivo. Este é um procedimento complexo, mas existem ferramentas disponíveis na *internet* que facilitam o processo. O acesso de *super* usuário dá controle total do sistema operacional *Android*, desbloqueando o acesso direto a áreas de memória do dispositivo e permitindo a cópia do arquivo de banco de dados, conforme é possível visualizar na Figura 7.

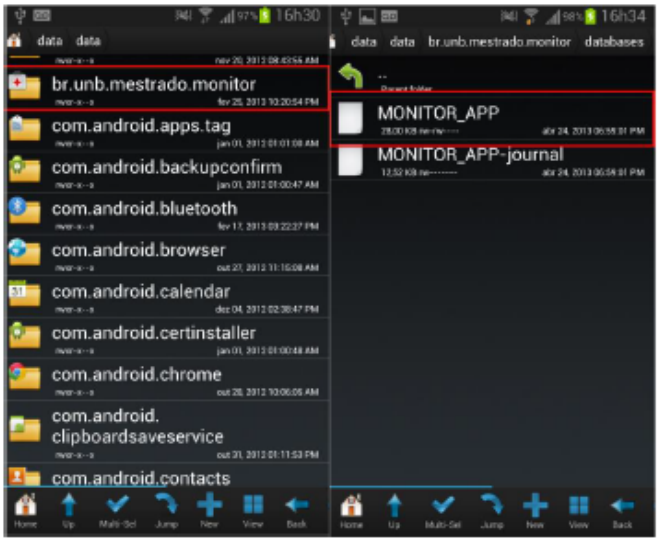

Figura 7 – Acesso ao arquivo de banco de dados através do sistema de arquivos com permissão de *super* usuário

A partir do momento em o arquivo de Banco de Dados esteja disponível, podem ser utilizadas ferramentas de acesso a base dados SQLite, como o SQLite Viewer for Android, para visualizar informações disponíveis na aplicação. Caso o atacante não conheça o nome do arquivo de banco de dados criado pelo *software* Monitor, uma das formas de descobri-lo seria observação da classe decompilada `br.unb.mestrado.monitor.persistence.MechanismSQLite.class`. Nas Figura 8 e Figura 9 é possível observar o acesso aos dados do arquivo de banco de dados, sem a necessidade de senha e sem nenhuma outra proteção.

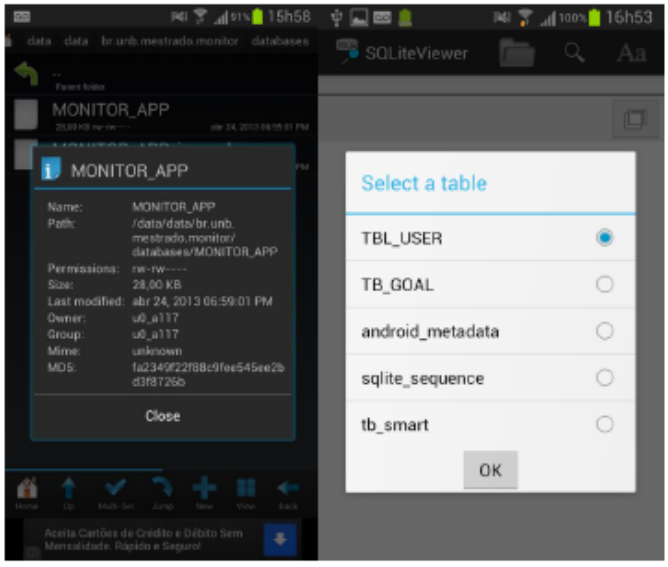

Figura 8 – Permissão do arquivo de banco de dados e acesso direto às informações do *software* Monitor

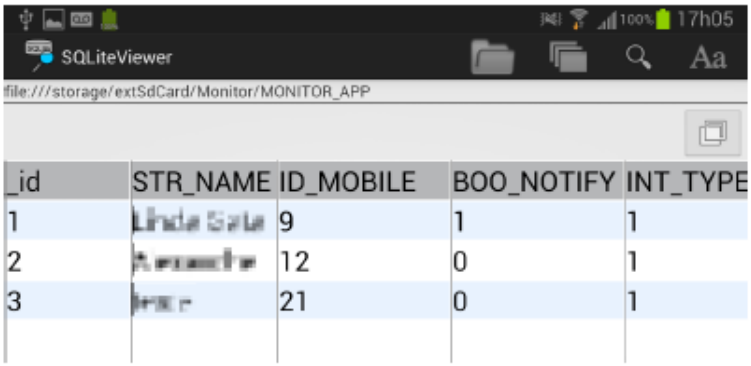

Figura 9 – Informações na tabela TBL_USER armazenadas pelo *software* Monitor

Validação insuficiente de dados de entrada

Descrição

O *software* Monitor não realiza a validação adequadamente de dados inseridos na sua aplicação. Por este motivo, é possível inserir dados no *software* Monitor que contenham caracteres utilizados em ataques de injeção de comando persistente. Este tipo de ataque pode levar a manipulação da informação o aplicativo Android e outras aplicações que venham a consumir as informações inseridas no *software* Monitor.

Ações Recomendadas

Construção de API para validação de dados de entrada inseridos no *software* Monitor contra caracteres especiais que possam levar a injeção de comando persistente.

Evidências

O *software* Monitor não permite que usuários sejam inseridos em sua base de dados se eles não estiverem nos contatos do dispositivo *Android*. Porém, não realiza validação da informação inserida, confiando no cadastro mantido pelo dispositivo do usuário.

Assim, um atacante pode cadastrar um contato no dispositivo *Android* que possua comandos a serem injetados na aplicação e inserir este contato (e o comando malicioso) no banco de dados do *software* Monitor, sem restrições, como pode ser visualizado na Figura 10.

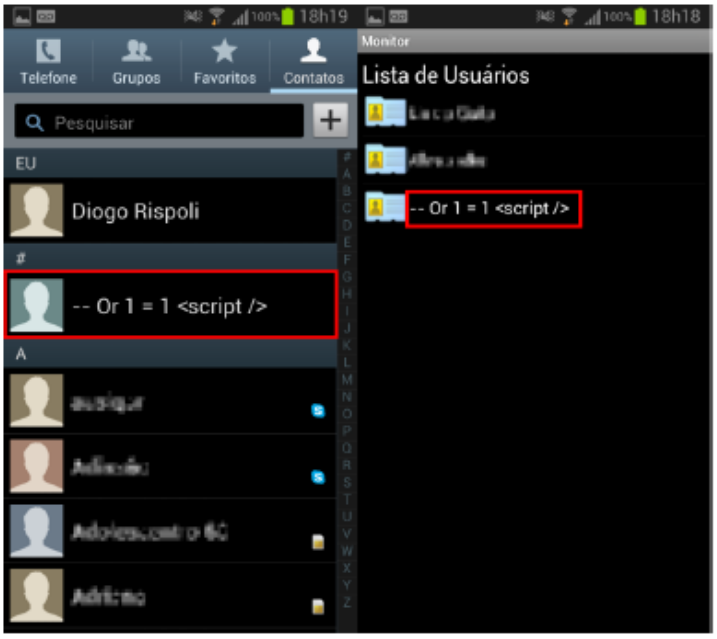

Figura 10 – Usuário com caracteres potencialmente perigosos no *software* Monitor

Durante a inserção do usuário, o *software* Monitor insere a informação de nome do contato em seu banco de dados conforme pode ser visualizado na Figura 11. Caso este banco de dados seja replicado / enviado para um servidor centralizado, estas informações ficarão persistidas e poderão ser consumidas por outras aplicações. O atacante pode aproveitar esta brecha para, por exemplo, realizar um ataque em uma pagina *web* de administração ou em aplicação de atendentes de *call-center*.

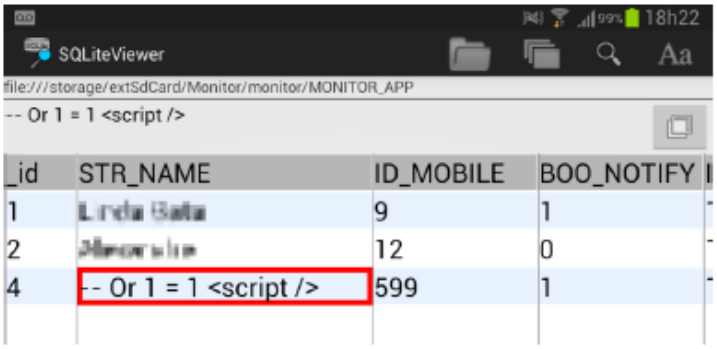

Figura 11 – Caracteres potencialmente perigosos persistidos no *software* Monitor

Anexo II – Requisitos de Segurança de Software

Os seguintes requisitos de segurança de *software* foram levados em consideração durante o teste de penetração:

- Autenticação segura;
- Transmissão de informação criptografada entre o sensor e o dispositivo *Android*;
- Proteção do arquivo da base de dados local;
- Validação de entrada contra injeção de comando;
- Alteração do código de conexão com o sensor;
- Armazenamento seguro de dados sensíveis;
- Armazenamento seguro de credenciais;
- Registro das atividades do usuário no *software* Monitor.

Anexo III – Casos de Abuso

Os casos de abuso se baseiam nos requisitos de segurança porque eles são um guia de como verificar se os requisitos de segurança foram implementados. Desta forma, os casos de abuso que foram definidos para o software Monitor, com uma breve descrição da exploração que será avaliada, são:

- Captura do tráfego entre o sensor e o dispositivo *Android*: O atacante obtêm as informações trocadas na conexão entre o sensor e o dispositivo *Android* para análise futura e roubo de informação;
- Vazamento de informação no *software* Monitor: O atacante obtêm informações através do acesso ao arquivo de banco de dados da aplicação armazenado no dispositivo *Android*.;
- Injeção de comando SQL: O atacante insere, modifica ou visualiza informações, de forma não autorizada, no banco de dados do *software* Monitor através da injeção de comandos SQL.;
- Manipulação não autorizada da informação transmitida: O atacante modifica informações, de forma não autorizada, através de ataque de *Man-in-the-middle* na conexão *bluetooth*.;
- Manipulação não autorizada da informação armazenada no banco de dados: O atacante modifica informações, de forma não autorizada, através de acesso direto ao arquivo do banco de dados do *software* Monitor.

ANEXOS

ANEXO 1: EXEMPLOS DE REQUISITOS DE SEGURANÇA DE SOFTWARE

Segue, apenas de forma ilustrativa, alguns exemplos de requisitos de segurança de *software*, retirados do livro *Official (ISC)2 Guide to the CSSLP* de Mano Paul.

Confidencialidade

- Senha e outros campos de entrada de dados sensíveis necessitam ser mascarados.
- Senhas não devem ser armazenadas as claras nos sistemas *backend*, e quando armazenadas devem passar por processo de *hash* com uma *função pelo menos equivalente a SHA-256.
- TLS (*Transport Layer Security)* como SSL (*Secure Socket Layer)* deve ser colocado em prática para proteger contra ameaças internas de *Man-in-the-Middle* (MITM) para todas as informações financeiras que sejam transmitidas.
- O uso de protocolos reconhecidamente inseguros como, por exemplo, FTP (*File Transfer Protocol*) para transmitir credenciais de contas em texto claro a terceiros fora de sua organização deve ser proibido.
- Arquivos de *log* não devem armazenar qualquer informação sensível como definido pelo negócio, de modo que seja compreensível por seres humanos.

Integridade

- Todos os formulários de entrada e *query strings* necessitam ser validadas frente a um conjunto de entradas aceitáveis, antes do *software* aceitá-los para processamento;
- O *software* a ser publicado deve ser disponibilizado juntamente com o *checksum* e a função *hash* usada pra computar o *checksum*, de modo que o interessado possa validar sua precisão e completude;
- Todos os personagens não humanos, como usuários para sistemas ou processos *batch*, devem ser identificados, monitorados e possuir escopo limitado no âmbito de sua alteração de dados, a medida de sua utilização nos sistemas que eles executam, a não ser que explicitamente autorizado para tal.

Disponibilidade

- O *software* deve oferecer alta disponibilidade de oito (8) noves (9), como definido pelo SLA (*Service Level Agreement*).
- O *software* deve estar preparado para atender capacidade máxima de 300 usuários simultâneos.
- O *software* e seus dados devem ser replicados por todos os centros de dados para prover balanceamento de carga e redundância.
- A funcionalidade de missão crítica no *software* deve ser restaurada a operação normal no prazo de 1 hora de descontinuidade; funcionalidade de missão essencial no *software* deve ser restaurada a operação normal no prazo de 4 horas da interrupção, e funcionalidade de missão suporte no *software* deve ser restaurada a operação normal no prazo de 24 horas.

Autenticação

- O *software* será implantado somente na *intranet* e o usuário autenticado deve fornecer novamente suas credenciais para acessar a aplicação, uma vez que esteja autenticado na rede.
- O *software* deverá suportar SSO (*Single-Sign-On*) a terceiros e fornecedores que estão definidos na lista de interessados.
- A política de autenticação garante a necessidade para dois – ou autenticação com múltiplos fatores para todo o *software* de processamento financeiro.

Autorização

- O acesso a arquivos secretos de alta sensibilidade deve ser restrito somente a usuários com níveis de permissão secreto e super-secreto.
- Os usuários não devem ser demandados a enviar suas credenciais sempre, uma vez que ele tenha se autenticado com sucesso.
- Todos os usuários autenticados herdarão a permissão de leitura somente que são parte do papel do usuário convidado enquanto os usuários autenticados por padrão terão permissão de leitura e escrita como parte do papel de usuário regular.

Somente os usuários com acesso administrativo, terão todos os direitos dos usuários regulares, adicionalmente a execução de operações.

Auditoria e *Logging*

- Todas as tentativas de autenticação devem ser registradas juntamente com o *timestamp* e o endereço de IP de origem da requisição.
- Os valores anterior e posterior a uma mudança de preço modificado por um usuário, quando da atualização de um preço por um usuário, devem ser monitorados com os seguintes campos auditados: identidade, ação, objeto e *timestamp*.
- Os *logs* de auditoria devem sempre ser adicionados de novos registros e nunca sobrescritos.
- Os *logs* de auditoria devem ser mantidos de forma segura por um período de 3 anos.

Gerenciamento de Sessão

- Cada atividade do usuário deverá ser rastreada de modo único.
- O *software* não deve solicitar as credenciais de acesso do usuário, uma vez que ele esteja autenticado na aplicação.
- As sessões devem ser explicitamente suspensas quando o usuário solicita o *logoff* ou fecha a janela do *browser*.
- Identificadores de sessão usados para identificar a sessão de usuários devem não ser passados em claro ou ser facilmente adivinhado.

Erros e Gerenciamento de Exceção

- Todos os erros e exceções devem ser explicitamente manipulados a partir de blocos *try, catch e finally*.
- Mensagens de erro que são mostradas ao usuário revelarão somente a informação necessária, sem vazamento de detalhes internos do sistema na mensagem de erro.
- Detalhes de exceções de segurança devem ser auditados e monitorados periodicamente.

Parâmetros de Configuração

- Os dados sensíveis do arquivo de configuração da aplicação web, como *strings* de conexão, devem ser encriptados.
- Senhas e chaves de criptografia não devem ser registradas no código fonte do *software*.
- A inicialização e a liberação de variáveis globais necessitam ser monitoradas com muito cuidado.
- Eventos de inicialização e interrupção de sessão devem incluir proteções na informação de configuração como uma salvaguarda contra ameaças de vazamento.

ANEXO 2: ACORDO DE CONFIDENCIALIDADE

ACORDO DE CONFIDENCIALIDADE

Termo de Compromisso quanto à Confidencialidade das informações decorrentes de avaliação de problemas de segurança da informação a ser realizado em aplicativo especifico. Esta avaliação é parte integrante de projeto acadêmico de pesquisa científica a ser realizado com o objetivo da obtenção do grau de mestre em engenharia biomédica e outras pesquisas correlatas.

Diogo de Carvalho Rispoli, brasileiro, Analista de Tecnologia da Informação, pessoa física, residente em STN Lote L Bloco D Apto 119, Brasília, DF, inscrito no CPF/MF sob o nº 726.862.001-20, doravante denominado **AVALIADOR,** e **Paula Gabriela de Medeiros Fernandes,** Analista de Tecnologia da Informação, pessoa física, residente em *Cond. Solar de Brasília, Quadra 01, Cj 06, Nro 20*, inscrita no CPF/MF sob o n.º *019.118.221-44*, doravante denominada **DESENVOLVEDORA**, resolvem celebrar o presente **TERMO DE CONFIDENCIALIDADE**, mediante as cláusulas e condições que seguem:

1.1. CLÁUSULA PRIMEIRA – DO OBJETO

O objeto deste Termo é prover a necessária e adequada proteção da **DESENVOLVEDORA** quanto ao tratamento e divulgação de informações confidenciais, sigilosas ou de acesso controlado, bem como códigos-fonte ou outras formas de propriedade intelectual restrita do qual o **AVALIADOR** venha a ter acesso, por qualquer meio, em razão de avaliação de segurança do *software* **MONITOR** que será realizada com objetivos acadêmicos e de pesquisa científica.

1.2. CLÁUSULA SEGUNDA - DAS INFORMAÇÕES CONFIDENCIAIS

1.2.1. Em função transferência de informações entre o **AVALIADOR** e a **DESENVOLVEDORA**, firma-se o presente Termo visando a resguardar as referidas partes de eventual má utilização ou repasse a terceiros não autorizados, sob pena de responder por suas responsabilidades nos termos da lei.

1.2.2. O **AVALIADOR** se obriga a manter o mais absoluto sigilo com relação a toda e qualquer informação que venha a ter acesso, que deverá ser tratada como informação sigilosa.

1.2.3. Deverá ser considerada como informação confidencial toda e qualquer informação escrita, verbal ou de qualquer outro modo apresentada, tangível ou intangível, podendo incluir, mas não se limitando a: regras de negócio, especificações, desenhos, cópias, diagramas, fórmulas, modelos, amostras, fluxogramas, croquis, fotografias, plantas, programas de computador, códigos-fonte, discos, processos, projetos, especificações, informações técnicas, financeiras, comerciais ou de patentes, dentre outros, doravante denominados "INFORMAÇÕES CONFIDENCIAIS", a que, diretamente ou indiretamente, o **AVALIADOR** venha a ter acesso ou conhecimento.

1.2.4. Compromete-se, outrossim, o **AVALIADOR** a não revelar, reproduzir, utilizar ou dar conhecimento, em hipótese alguma, a terceiros, bem como permitir que seus parceiros de pesquisa façam uso dessas INFORMAÇÕES CONFIDENCIAIS de forma diversa ao estritamente necessário para resposta a pesquisa cientifica a ser realizada.

1.2.5. O **AVALIADOR** deverá cuidar para que as INFORMAÇÕES CONFIDENCIAIS fiquem restritas ao conhecimento de pessoas que estejam diretamente envolvidos nas discussões, análises e reuniões, devendo dar-lhes ciência da existência deste Termo e da natureza confidencial destas informações.

1.3. CLÁUSULA TERCEIRA – DAS LIMITAÇÕES DA CONFIDENCIALIDADE

1.3.1. As estipulações e obrigações constantes do presente instrumento não serão aplicadas a nenhuma informação que:

1.3.2. Seja comprovadamente de domínio público, exceto se isso ocorrer em decorrência de ato ou omissão do avaliador;

1.3.3. Já esteja em poder do **AVALIADOR** como resultado de sua própria pesquisa ou desenvolvimento interno, contanto que o **AVALIADOR** possa comprovar esse fato;

Paula G M Fernandes

1.3.4. Tenha sido comprovada e legitimamente recebida de terceiros, estranhos, de toda forma, ao presente Termo.

1.3.5. Seja revelada em razão de requisição judicial ou outra determinação válida do Governo, somente até a extensão de tais ordens, desde que o **AVALIADOR** cumpra qualquer medida de proteção pertinente e tenha notificado a existência de tal ordem, previamente e por escrito, a desenvolvedora, dando a esta, na medida do possível, tempo hábil para pleitear medidas de proteção que julgar cabíveis.

1.4. CLÁUSULA QUARTA – DOS DIREITOS E OBRIGAÇÕES

1.4.1. O **AVALIADOR** se compromete e se obriga a utilizar toda e qualquer INFORMAÇÃO CONFIDENCIAL exclusivamente para os propósitos deste Termo e do projeto de pesquisa em curso, mantendo sempre estrito sigilo acerca de tais informações.

1.4.2. O **AVALIADOR** se compromete a não efetuar qualquer cópia da INFORMAÇÃO CONFIDENCIAL sem o consentimento prévio e expresso da **DESENVOLVEDORA**.

1.4.3. O consentimento mencionado no item 1.4.2 supra, entretanto, será dispensado para cópias, reproduções ou duplicações para os fins acima referidos, pelos parceiros do avaliador que necessitem conhecer tal informação ou que participem do projeto de pesquisa.

1.4.4. O **AVALIADOR** compromete-se a cientificar seus parceiros e/ou prepostos da existência deste Termo e da natureza confidencial das informações.

1.4.5. O **AVALIADOR** obriga-se a tomar todas as medidas necessárias à proteção da INFORMAÇÃO CONFIDENCIAL da **DESENVOLVEDORA**, bem como para evitar e prevenir sua revelação a terceiros, exceto se devidamente autorizado por escrito pela **desenvolvedora**.

1.4.6. O **AVALIADOR** tomará as medidas de cautela cabíveis, na mesma proporção em que tomaria para proteger suas próprias informações confidenciais, a fim de manter as informações confidenciais em sigilo.

1.4.7. O **AVALIADOR** deverá firmar acordos por escrito com seus parceiros de pesquisa cientifica, cujos termos sejam suficientes a garantir o cumprimento de todas as disposições do presente Termo.

1.4.8. O presente Termo não implica a concessão pela **DESENVOLVEDORA** ao **AVALIADOR** de nenhuma licença ou qualquer outro direito, explícito ou implícito, em relação a qualquer direito de patente, direito de edição ou qualquer outro direito relativo à propriedade intelectual.

1.4.9. O **AVALIADOR** obriga-se a não tomar qualquer medida com vistas a obter, para si ou para terceiros, os direitos de propriedade intelectual relativos às INFORMAÇÕES CONFIDENCIAIS que venha a ter conhecimento.

1.4.10. O **AVALIADOR** compromete-se a separar as INFORMAÇÕES CONFIDENCIAIS dos materiais confidenciais de terceiros para evitar que se misturem.

1.5. CLÁUSULA QUINTA – DO RETORNO DE INFORMAÇÕES CONFIDENCIAIS

1.5.1. Todas as INFORMAÇÕES CONFIDENCIAIS que o **AVALIADOR** venha a tomar conhecimento permanecem como propriedade exclusiva da **DESENVOLVEDORA**, devendo a esta retornar imediatamente assim que por ela requerido, bem como todas e quaisquer cópias eventualmente existentes.

1.6. CLÁUSULA SEXTA – DA VIGÊNCIA

1.6.1. O presente Termo tem natureza irrevogável e irretratável, permanecendo em vigor perpetuamente após o término do projeto de pesquisa cientifica mencionado, ao qual este é vinculado.

1.7. CLÁUSULA SÉTIMA – DAS PENALIDADES

1.7.1. A violação de qualquer das obrigações mencionadas neste instrumento sujeitará o **AVALIADOR** à aplicação das penalidades cabíveis, cíveis e criminais, nos termos da lei, obrigando-a ainda a indenizar a **DESENVOLVEDORA** a todo e qualquer dano, perda ou prejuízo.

1.8. **CLÁUSULA OITAVA - DAS DISPOSIÇÕES GERAIS**

1.8.1. A **DESENVOLVEDORA** poderá, ainda, propor qualquer medida, administrativa ou judicial, para impedir ou invalidar tais violações.

1.8.2. Este documento constitui termo vinculado ao projeto de pesquisa cientifica com o objetivo da obtenção do grau de mestre em engenharia biomédica e outras pesquisas correlatas, como parte independente e regulatória daquele.

1.8.3. Surgindo divergências quanto à interpretação do pactuado neste Termo ou quanto à execução das obrigações dele decorrentes ou, ainda, constatando-se nele a existência de lacunas, solucionarão as partes tais divergências de acordo com os princípios de boa fé, da equidade, da razoabilidade e da economicidade, preenchendo as lacunas com estipulações que, presumivelmente, teriam correspondido à vontade das partes na respectiva ocasião.

1.8.4. O disposto no presente Termo prevalecerá, sempre, em caso de dúvida, e salvo expressa determinação em contrário, sobre eventuais disposições constantes de outros instrumentos conexos firmados entre as partes quanto ao sigilo de informações confidenciais, tal como aqui definida.

E, assim, por considerarem justas as condições, as partes assinam o presente instrumento em duas vias de igual teor e um só efeito.

Brasília, DF, 8 de fevereiro de 2013.

Como **AVALIADOR**:

Diogo de Carvalho Rispoli

Como **DESENVOLVEDORA**:

Paula Gabriela de Medeiros Fernandes

Paula Gabriela de Medeiros Fernandes

ANEXO 3: PUBLICAÇÕES

QUALIDADE DO SOFTWARE PARA EQUIPAMENTOS MÉDICOS: UMA NOVA PERSPECTIVA

Diogo C. Rispoli*, Vinicius C. Rispoli*, Carolina G. Alves* e Lourdes M. Brasil*

*Programa de Pós-Graduação em Engenharia Biomédica, Faculdade do Gama/UnB, Brasília, Brasil

drispoli@gmail.com

Abstract: Despite the widespread use of information technology in the medical area, little attention is given to software security that performs, or collects information from patient within medical devices. In order to improve the quality and safety of these devices, we propose in this work other perspective for construction and analysis of software/firmware that accompany those devices. This assessment will undergo consultation existing standards, analysis of software safety techniques and their alignment with the medical area and inspection for problems in software built for this purpose.

***Palavras-chave*: Segurança da informação, *Software* seguro, *Hackers*, Ciclo de vida do *software* médico, Qualidade do *software* para dispositivos médicos.**

Introdução

Em um cenário de constante evolução tecnológica, a busca por soluções que atendam a área médica é uma constante. Atualmente é possível encontrar *software* disponíveis gratuitamente pela *internet* que coletam informações sobre a saúde dos indivíduos e que podem ser instalados em dispositivos móveis, como *smartphones e tablets* [1]. Não obstante desta velocidade, as normas voltadas para a área médica não disponibilizam modelos em que seja possível avaliar os problemas associados a vulnerabilidades comuns que podem aparecer num ciclo de desenvolvimento de *software*.

Normas de fato como a ISO/IEC 62304:2007 [2] que tratam do desenvolvimento do *software* para equipamentos médicos, apesar de atuais, não lidam com o problema da universalização deste tipo de serviço. Assim como a norma ISO 27799:2008 [3] que lida com problemas relacionados à segurança dos sistemas de informação médica, mas não endereça soluções relacionadas ao desenvolvimento de *software* seguro, face ao momento atual.

Existe, inclusive, uma preocupação dos consumidores e profissionais da área de saúde a respeito dos sistemas de informação inteligentes que podem ser utilizados fora do consultório médico. Os participantes da pesquisa [4] informaram o desejo de dividir os dados coletados por dispositivos de saúde ubíquos com as pessoas ou instituições responsáveis por acompanhar a sua saúde, mas temem que estas informações possam ser acessadas de forma indiscriminada por indivíduos ou grupos que desejem manipular inadequadamente estas informações.

Evidentemente, é premente a adoção de padrões e ações que garantam a segurança do *software* presente nos dispositivos médicos. A exploração de vulnerabilidades em equipamentos médicos podem levar a morte ou lesões graves, fraudes, revelação não autorizada de informação, usurpação, entre outros ataques. Por este motivo, deve-se garantir que requisitos de segurança da informação (integralidade, confidencialidade, disponibilidade e não repúdio da informação coletada) sejam cumpridos, bem como assegurar que vulnerabilidades de *software* não sejam inseridas nestes dispositivos durante o seu ciclo de desenvolvimento.

Este artigo abordará uma visão para a melhoria da segurança de aplicações médicas a partir da avaliação de riscos à segurança da informação e da correlação de requisitos para segurança do *software* (e suas técnicas de mitigação) com os padrões relacionados à segurança no suporte à vida. O resultado desta correlação serão atividades propostas para o ciclo de vida do *software* presente em dispositivos médicos.

Materiais e Métodos

Cenário Atual – O momento de efervescência tecnológica atual abre portas para que *hackers* explorem e promovam usurpações como o ataque a uma bomba de insulina promovido, documentado e apresentado em [4]. Baseado em uma técnica simples, que alia conhecimentos de programação e eletrônica básica, o *hacker* empreende um ataque a uma bomba de insulina de seu próprio uso com o objetivo de demonstrar a perspectiva inocente com que estes equipamentos são construídos. Como resultado final, ele consegue aplicar uma dose letal de insulina quebrando a segurança de autenticação imposta pela comunicação sem fio do equipamento.

Vale lembrar que este não foi o primeiro caso de ataque documentado a equipamentos médicos. Em 2008, uma equipe norte-americana de estudiosos publicou um artigo em que mostra um ataque a um marca-passo/desfibrilador, que expunha também falhas de segurança relacionadas à conexão sem fio do equipamento [5].

Falhas no *Software* para Equipamentos Médicos – Com sua gama intrínseca de componentes, os equipamentos médicos estão sujeitos a partes com problemas. Dentro destas diversas partes, identificar e quantificar no *software* os potenciais efeitos de falhas e defeitos é complexo. Vários dispositivos compartilham componentes eletrônicos e mecânicos em comum e seus processos de busca, correção e documentação de falhas já estão bem consolidados.

Por outro lado, geralmente o *software* é construído sob demanda e para ser utilizado em dispositivos específicos. Com raras exceções, não existem mecanismos efetivos para estabelecer e mapear falhas ou problemas no *software*. Como consequência, recai sobre os fabricantes a responsabilidade de comprovar que o *software* para equipamentos médicos são seguros e efetivos [6].

Ciclo de vida do *software* médico - Um modelo de ciclo de vida organiza as atividades de desenvolvimento de *software* e provê um *framework* para monitorar e controlar o seu projeto de construção e operação. Sem a adoção de um modelo é difícil endereçar em que momento o desenvolvimento e a validação do projeto se encontram e até mesmo como e em que situações os pontos de controle devem ser aplicados [7, 8].

Apesar das QSRs (*Quality Systems Regulations*) não estabelecerem um modelo específico de ciclo de vida para o *software* médico, as normas regulatórias colocam que a adoção de um modelo é importante e que este deve conter pelos menos as fases de planejamento da qualidade, definição de requisitos, especificação do projeto de *software*, construção (ou codificação), teste, instalação, operação e suporte, manutenção e retirada de circulação [7, 9].

O ciclo de vida faz parte dos controles do projeto e estes controles são necessários para reduzir a probabilidade de inserção de defeitos no dispositivo médico [7]. Então, para mitigar o problema das vulnerabilidades dentro do *software* médico, é essencial indicar atividades a serem aplicadas entre as fases do ciclo de vida das aplicações. Estas atividades estão relacionadas à identificação, construção e validação de técnicas que impossibilitem a exploração de vulnerabilidades na operação do *software*.

Resultados Parciais

Uma forma de associar questões que aparentemente estão desconectadas é observar como aspectos de segurança de *software* estão ligados a construção de dispositivos médicos seguros. Sob o olhar do *software*, os riscos são caminhos através da aplicação por onde atacantes podem prejudicar o negócio ou as organizações [10]. Observando-se o mesmo aspecto pela perspectiva do equipamento, o risco (ou nível de preocupação) é uma estimativa da severidade da lesão que um equipamento pode permitir ou infligir, direta ou indiretamente, em um paciente ou operador como resultado de falhas do dispositivo, falhas de projeto, ou simplesmente, em virtude do emprego do dispositivo para seu uso pretendido [9]. Objetivamente, estas duas visões se aproximam, pois o risco está diretamente ligado a falhas do *software* ou fraquezas em seus mecanismos de controle. Sua consequência natural é a subversão e danos de diversas naturezas (primariamente danos à vida, mas também financeiro e da imagem corporativa dos fabricantes) provocados por pessoas mal intencionadas, como pode ser visualizado na Figura 1.

Figura 1: Riscos de Segurança da Aplicação [10].

Sob esta perspectiva levantar, mapear e balancear os riscos, falhas e vulnerabilidades introduzidas por problemas na construção do *software* médico torna-se uma tarefa exaustiva e pouco efetiva, além de afastada da realidade tecnológica atual. Existem diversos padrões, conforme é possível observar na Figura 2, que tratam direta (como a norma ISO/IEC 62304:2006) ou indiretamente do ciclo de desenvolvimento de *software* e os seus riscos associados [11]. Entretanto, existe uma carência de metodologias que abordem aspectos ligados à mitigação de vulnerabilidades exploráveis em *software*.

A norma ISO 27799:2008, em sua preocupação relacionada aos sistemas de informação médica, não trata ou endereça soluções relacionadas ao processo de construção de *software* seguro. Apesar disto, coloca requisitos para a operação de sistemas de informação seguros como autenticação, autorização, responsabilização, utilização de criptografia, transporte seguro da informação e proteção contra códigos móveis que são aspectos relevantes em que o *software* é o ator principal ou atua como coadjuvante importante [3].

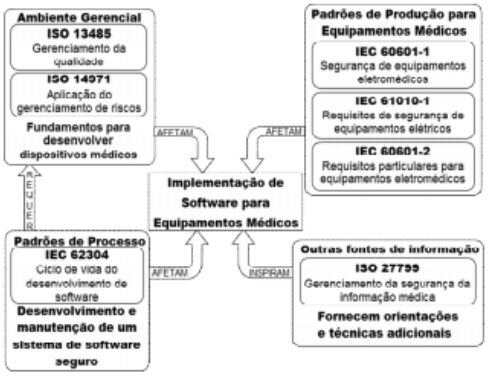

Figura 2: Norma ISO/IEC 63304 e seus relacionamentos com outros padrões [11].

Observando os processos de garantia da qualidade empregados para as aplicações médicas, atualmente, os aspectos de validação e verificação estão somente preocupados com requisitos funcionais do *software* [10].

Então, é necessário tratar e listar interações com o ciclo de vida do *software* que orientam como realizar testes de penetração ou auditorias, levantar requisitos não funcionais para implantação e operação segura de aplicações, incluir a análise de risco de vulnerabilidades no projeto do *software*, proteger as aplicações contra falhas de injeção de comando e *buffer overflow*, tratar corretamente erros e exceções e responsabilizar gravando registros sanitizados (i.e., após remoção de informações sensíveis) da atividade dos usuários na operação do equipamento [3, 8, 10, 12].

Também é importante projetar mecanismos eficientes de autenticação, gerenciamento de sessão segura, autorização do usuário autenticado, criptografia para o transporte e armazenamento seguro das informações coletadas e dos registros dos pacientes com o objetivo de aumentar a garantia da segurança e da qualidade dos sistemas de informação de saúde e suas aplicações correlatas (que tem como ativos os dispositivos e seu *software* associado) [3, 10].

Discussão e Conclusão

Com a adoção de padrões de construção de *software* seguro dentro do seu ciclo de vida, a partir de atividades que garantam a identificação, avaliação, tratamento, aplicação e validação de controles de segurança da informação, espera-se o aumento da qualidade e diminuição máxima da superfície de ataque explorável dentro das aplicações médicas. O levantamento e o desenvolvimento de *checklists* com os controles a serem aplicados pode auxiliar a incorporação de práticas de codificação defensiva ao longo da construção das aplicações médicas.

O tratamento dos aspectos relacionados a segurança do *software* não necessariamente representa aumento do custo no seu ciclo de vida de desenvolvimento, tendo em vista que corrigir problemas e falhas desta natureza custam mais depois da aplicação pronta e em produção [8].

É importante também que a responsabilidade de avaliar os aspectos de segurança dentro do *software* produzido e utilizado pelos dispositivos médicos faça parte dos programas de verificação e validação adotados pelos organismos designados para tal. Órgãos como a ANVISA (Agência Nacional de Vigilância Sanitária) no Brasil podem instituir dentro do seu modelo de validação e verificação de qualidade de *software* análises estáticas e dinâmicas, empreendidas de forma manual ou automatizada, com o objetivo de analisar se o *software* que acompanha os dispositivos médicos é robusto na capacidade de resistir a ataques promovidos por *hackers*.

É necessário modificar a perspectiva com que as aplicações voltadas para dispositivos médicos são construídas. Buscar novos parâmetros mesclando os requisitos de segurança para aplicações com os requisitos de segurança à vida deve ser uma constante a ser tratada com cuidado. Falhas de *software* são consequências naturais da análise, projeto, e construção das aplicações, porém uma visão inocente pode diminuir a capacidade crítica de observar que falhas são aspectos que vão além dos defeitos (*bugs*) e que o risco relacionado às vulnerabilidades de segurança não pode ser ignorado.

Como uma segunda etapa deste trabalho, dá-se a criação de atividades que podem ser associadas ao ciclo de vida do *software* médico. Reunir padrões, normas técnicas, metodologias e boas práticas de desenvolvimento seguro em um compêndio podem minimizar os riscos e falhas associados a segurança das aplicações. Como forma de avaliar as atividades inseridas dentro do ciclo de vida do *software*, pode-se realizar a validação e verificação do *software* de um equipamento médico utilizando os parâmetros propostos.

Referências

[1] IBM Institute for Business Value (2011), *The future of connected health devices*. Disponível em: http://www-935.ibm.com/services/us/gbs/thoughtleadership/ibv-connected-health-devices.html. Acesso em 01 jun. 2012.

[2] IEC 62304 (2006), *Medical device software – Software life cycle processes*, 1ª ed, Geneva.

[3] ISO 27799 (2008), *Health informatics – Information security management in health using ISO/IEC 27002*, 1ª ed, Geneva.

[4] Radcliffe, J. (2011), "Hacking Medical Devices for Fun and Insulin: Breaking the Human SCADA System", In: *Black Hat Conference*. Disponível em https://media.blackhat.com/bh-us-11/Radcliffe/BH_US_11_Radcliffe_Hacking_Medical_Devices_WP.pdf. Acesso em 01 jun. 2012.

[5] Halperin, D. et al (2008), "Pacemakers and Implantable Cardiac Defibrillators: Software Radio Attacks and Zero-Power Defenses" In: *Proceedings of the IEEE Symposium on Security and Privacy 2008*.

[6] Rakitin, S. R. (2006), "Coping with Defective Software in Medical Devices" In: *Computer Magazine - IEEE Computer Society*, v. 39, n. 4, p. 40-45.

[7] Vogel, A. D. (2011), *Medical Device Software Verification, Validation, and Compliance*, Boston: Artech House.

[8] McGraw, G. (2006), *Software security: building security in*, Boston: Addison Wesley Professional.

[9] FDA (2005), *Guidance for the Content of Premarket Submissions for Software Contained in Medical Devices*. 1ª ed, New Hampshire.

[10] Open Web Application Security Project (2010), *OWASP Top Ten Guide*. Disponível em: https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project. Acesso em 10 jun. 2012.

[11] Hall, K. (2010), "Developing Medical Device Software to IEC 62304", In: *European Medical Device Technology*, v. 1, n. 6.

[12] Anderson, R. J. (2008), *Security Engineering: A Guide to Building Dependable Distributed Systems*, 2ªed, Indianapolis: Wiley Publishing, inc.

Software Lifecycle Activities to Improve Security Into Medical Device Applications

Diogo C. Rispoli , Lourdes M. Brasil
Graduate in Biomedical Engineering
University of Brasília at Gama
Brasília, Brazil
e-mail: drispoli@gmail.com, lmbrasil@gmail.com

Vinicius C. Rispoli
Faculty of Engineering at Gama
University of Brasília
Brasília, Brazil
e-mail: vrispoli@unb.br

Paula G. Fernandes
Computer Science Department
University of Brasilia
Brasília, Brazil
e-mail: paulag6@gmail.com

***Abstract*—This work proposes a methodology to include into Medical Software Development Lifecycle activities that helps improve security. The methodology uses assessment techniques and methods, applied to each phase of software lifecycle, that address security concerns and help to improve software quality. As a result, a partial analysis using the methodology proposed was performed in medical software at development stage to help reduce its gap between safety and security requirements.**

Keywords-information security; security software; hackers; medical software lifecycle; security risks.

I. INTRODUCTION

In a scenario of constant technological evolution, the demand for solutions in medical field is constant. Nowadays, you can find free software available on the internet which collect vital information of individuals and can be installed on mobile devices, such as smartphones and tablet computers [1]. Despite this speed, standards focused on medical field do not present models that make possible to assess problems associated with common security vulnerabilities that may appear in the software development cycle.

Standards as ISO / IEC 62304:2007 [2] dealing with the medical equipment software development, although recent, do not handle with these new technological perspectives. On the other hand, ISO 27799:2008 [3] deals with security concerns of health information systems, but it does not address solutions related to secure software development, compared to the present moment.

This very moment of technology effervescence opens doors for hackers to exploit and promote invasions as the attack on an insulin pump documented and presented in [4]. Based on a simple technique that combines programming skills and basic electronics, the hacker undertakes an attack on an insulin pump, used by himself, in order to demonstrate the innocent perspective that these devices are built. As a final result, he can apply a lethal dosage of insulin breaking the authentication security required by the equipment wireless communication.

Remembering that this was not the first case of attack documented on medical devices. In 2008, a U.S. team of researchers published a paper that showed an attack on a pacemaker, which also exposed security flaws related to wireless equipments [5].

Evidently, there is a rush in adopting standards, and actions, to ensure the security of the software built for medical devices. The exploitation of vulnerabilities in medical equipment can lead to death or serious injury, fraud, unauthorized disclosure of information, theft, and other attacks. For this reason, it is necessary to ensure that information security requirements (integrity, confidentiality, availability and non-repudiation of data collected) are met as well as ensuring that software vulnerabilities are not included in these devices during its development lifecycle.

This article will discuss requirements for improving the security of medical applications based on risk assessment of information security and the correlation of requirements for software security standards and their mitigation techniques related to safety in life support. As a result of this work, activities, also known as touch points, will be shown and assessed through a software development lifecycle helping to ensure the security requirements needed to consider software secure and safe.

This document is divided in six sections. In the next section, the relationship between risk perspective from safety and security views will be discussed. In section three, we will present the importance of software lifecycle and the incorporation of security activities into software construction. In the fourth section, the assessed software and its characteristics will be presented. Software assessment against the methodology proposes will be shown the in fifth section. And, in last section, will be discussed the assessment results.

II. SECURITY RISKS

In order to associate issues that are seemingly disconnected, it is important to observe how software security aspects are linked to medical devices construction. Under the perspective of the software, the risks are paths through the application where attackers (hackers) can disrupt business or organizations [10].

Observing this look from the perspective of the equipment, the risk (or level of concern) is an estimate of the injury severity which equipment can inflict or allow, directly or indirectly, in a patient or operator, as a result of device failure, design flaws, or because of the device employment for its intended use [9].

 ISBN: 978-1-61208-250-9

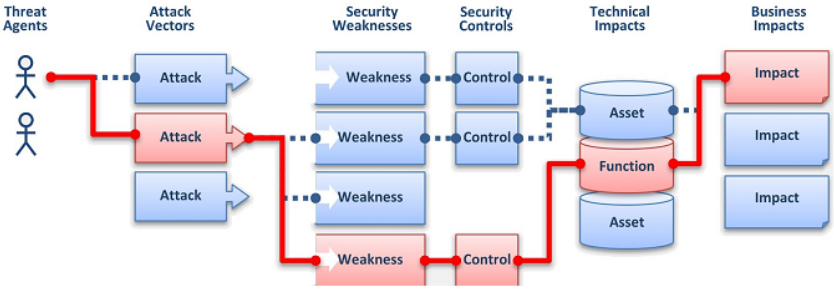

Figure 1. Applications Security Risks [10].

Objectively, these two views are very close because risks are directly related to software failures or weaknesses in its control mechanisms. Its natural consequence is the subversion and many kinds of damage, primarily damage to life, but also financial and corporate image caused by malicious people.

Threat agents can use several paths over application in order to attack organizations. These paths are through exploration of security weaknesses to bypass security controls and cause technical and business impacts, as it can be seen in Figure 1.

From this perspective, raising, mapping and balancing risks, flaws and vulnerabilities introduced by problems in the construction of medical software becomes exhausting, ineffective and away from the current technological reality. There are many patterns as you can see in Figure 2, dealing directly (such as IEC 62304:2006) or indirectly with software development lifecycle and its associated security risks [11].

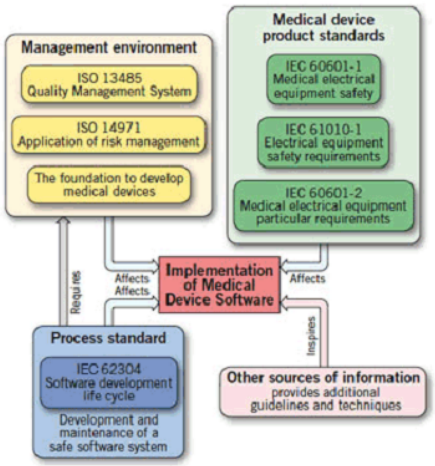

Figure 2. ISO/IEC 63304 and its relation to other standards [11].

However, there is a lack of methodologies that address mitigation aspects for exploitable vulnerabilities in software. It is important notice that IEC 62304:2006 address security as concern that manufacturers shall include in software requirements [2].

The ISO 27799:2008, which concerns to medical information systems, do not treat or address solutions related to the process of building secure software. Notwithstanding, this standard imposes requirements on the operation of informational systems as secure authentication, authorization, accountability, use of encryption, secure information communication and protection against code injection. These are relevant aspects where the software is the leading actor or an important supporting actor [3].

Observing the processes of quality assurance employed in medical applications, the aspects of validation and verification are only concerned with functional requirements of the software [10].

So, it is necessary list interactions with the lifecycle of the software that shows a path to perform penetration tests and audits, raise non-functional requirements for safe operation and deployment of applications, including risk analysis of vulnerabilities in software design, protect applications against command injection flaws and buffer overflow, properly handle errors and exceptions and logging sanitized records (after removing sensitive information) of users activity in the equipment operation [3, 8, 10, 12].

It is also important to design efficient mechanisms for authentication, secure session management, user authorization, authenticated encryption for secure transmission, storage of collected data and records of patients with the goal of increasing the guarantee of the safety and quality of information systems health and its related applications, whose assets are devices and their associated software [3, 10].

III. SOFTWARE DEVELOPMENT LIFECYCLE AND SECURITY CONCERNS

Lifecycle models organize development software activities and provide a framework to monitor and control a building software project and its future operation. Without a

 ISBN: 978-1-61208-250-9

model is difficult to say the exact moment of project's development or validation phase and how or which situations control activities must be applied [7, 8].

Despite *Quality Systems Regulations (QSRs)* do not establish a specific lifecycle model for medical software, regulatory standards state that a model adoption is important and it should contain at least some phases like quality planning, requirements management, software project specification, coding, testing, installation, operation, support and maintenance [7, 9].

Development of checklists with controls to be applied can aid incorporation of secure coding practices throughout the construction of medical software. The use of security software techniques does not necessarily increase the cost of its development lifecycle, in order to correct problems and failures of this nature cost more after application development finishes [8].

Adoption of security techniques is expected since medical software is able to run into smartphones and other mobile devices, for example, and all information collected and transmitted by those devices are sensitive and confidential.

Software development lifecycle is part of project controls and these controls are needed to reduce flaws insertion in medical device [7]. So, to help mitigate medical software vulnerabilities problem is essential to indicate what activities must be implemented between software lifecycle phases. These activities are related to the identification, development and validation of techniques that difficult vulnerabilities exploitation in software operation.

An interesting way to improve software security and quality is perform security activities through software lifecycle. Those activities are responsible to manage security concerns and must be applied inside lifecycle phases instead deal with security concerns only at requirement phase, as suggested by IEC 62304:2006 [2]. A correlation between phases and activities can be seen in Figure 3; they were described for general software projects in [8].

It is important to notice that there is no specific methodology to use the described security touch points. They can be applied in every kind of software development lifecycle methodology [8].

A. A brief description of each touch point

The touch points are described as follows [8]:

Abuse Cases – Build abuse cases is relevant to do relationship between problems and risk analysis. At that moment is important observe if some attack pattern fits the system or software requirements. This is a good moment to model vulnerability scenarios that could be exploited in Code Review Phase and Penetration Testing Phase.

Security Requirements – Security requirements must cover functional security, safety requirements, raised abuse cases and attack patterns. In that phase every software security necessity must be mapped to ensure the correct implementation. A good example for security requirements is the correct use of cryptography to protect critical data.

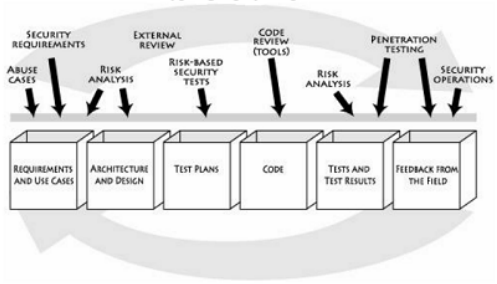

Figure 3. Security touch points inside a lifecycle [8].

Architectural Risk Analysis – Completing risk analysis oriented by ISO 14971. This analysis is a small part of a Risk Management Process that every Manufacturer must apply complying with ISO 14971, according to [2].

Risk Based Security Tests – The testing strategy must cover at least to major topics: test security requirements with standard functional testing techniques and risk-based security testing build from abuses cases and attack patterns.

Code Review – After codification, and before testing phase, the code review analysis is a good activity to ensure the security requirements were well implemented and the vulnerabilities listed in abuses cases analysis are outside the software. The code review can be automatic or manual and each strategy has pros and cons. Automated tools do not enforces all scenarios; some will require manual assessment [14].

Penetration Testing – This is a set of techniques and tools used together to test the software application dynamically against design flaws or vulnerabilities. This activity is important to guarantee that the software or its infrastructure do not have any potential problem that can be exploited in a particular way and change its behavior on the fly.

Security Operations – It is very important to log the user activity into software system usage. Even more important is to maintain that data in a correct and protected manner, to ensure that the attacker or attack activities can be tracked down after the attack attempt.

IV. SOFTWARE ASSESSED

The assessed medical device is responsible to monitor vital signs from a patient and send collected information to an Android smartphone. This system, showed as a diagram at Figure 4, is divided into a Body Sensor Network (Figure 5a), composed by a sensor set that monitor vital signs, a Coordinator sensor that collects information from body sensors in a regular basis and re-send that data to the smartphone and the Monitor software (Figure 5b) that evaluate patient conditions time to time. For this software /

 ISBN: 978-1-61208-250-9

equipment no injury is possible, arranging it into Class A classification, according to [2].

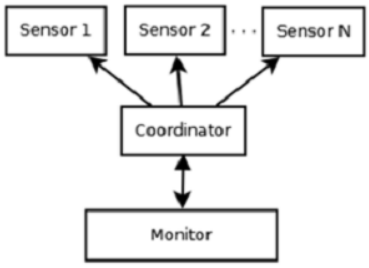

Figure 4. Monitor System Diagram.

This medical device is developed as a research project of the Software Engineer Group from Computer Science Department at University of Brasilia and was provided as a courtesy for this assessment. The research group responsible for developing the monitor system is not the same group that performed the software analysis. Notice that the only part assessed in this work is the Monitor software. Mechanical parts, sensors and smartphone hardware are not part of that analysis. Monitor software was developed in Java Language to run in Android devices.

This software uses the *Software Product Line (SPL)* methodology to build reusable components. In *SPL*, each product is a different piece of software that has some common artifacts in its structure [17]. In medical area, the use of *SPL* methodology brings some problematic issues related to validation and verification of safety characteristics. So, the research team [17] built Monitor software to verify the use of a parametric validation checking model to ensure safety properties (availability, reliability, security, integrity and maintainability). It was done because all medical device software must have dependable and reliable characteristics to guarantee safety.

This device and its related software were a good candidate to security evaluation since the software was in early development state and uses an unusual development methodology for medical devices. It is especially interesting to see if security activities really fit into a new development methodology or perspective.

V. Analysis of the Software Based on the Methodology Proposed

The analyzed software was not plan or built with any security touch point in mind. To help improve the software security and safety was performed an evaluation to propose and add touch point activities into software lifecycle, especially into building steps. Those touch points could be added into software lifecycle at any time, but it is better to do it when the software contains those activities from the scratch.

There are some steps to assessment take place. These steps can be related with one or more touch points each time and they were performed to track the assessed software into a security lifecycle.

Just for the record, safety practices listed at ISO 62304:2006 and other standards will not be ignored here but overlapped by security perspectives. It will be added at software process to increase safety and establish security. For example, risk analysis, abuse cases and risk-based tests are already present in safety related processes and this work will bring security concerns to these activities.

Abuse cases are related with vulnerabilities and flaws. For this analysis were defined SQL injection vulnerability and authentication and authorization problems as abuse cases. SQL injection, for example, could reveal validation problems in application. That is a common vulnerability in software [10, 14, 15] and must be mitigated. Authentication and authorization problems could show problems related to software design flaws [15].

Risks, in a security perspective, are directly related to software failures or vulnerabilities. The risk for SQL injection vulnerability is information disclosure and for authentication and authorization problems are non-legitimate user accessing and exploring the application. The risk-based security tests will be related to the risks specified. In code phase, these risks must be mitigated to ensure no path for exploitation.

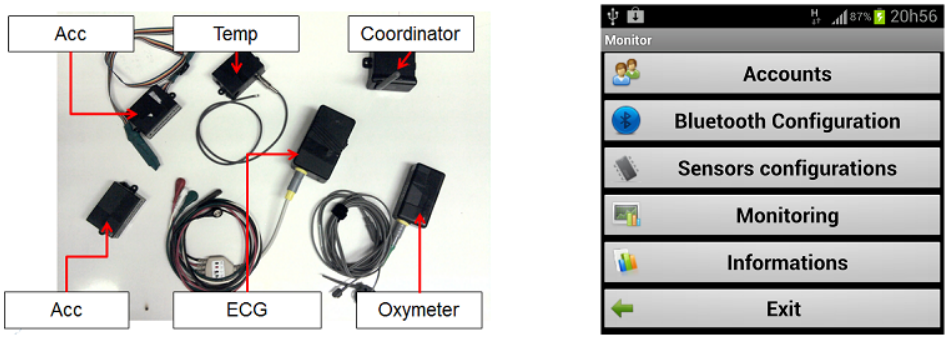

Figure 5. (a) Body Sensor Network (*left*). (b) Monitor software interface (*right*).

 ISBN: 978-1-61208-250-9

Code review is an important control strategy. This methodology comprises, at least, the following elements: Track user-controllable entry point data and review source code responsible for process it, search evidences to ensure that there is no vulnerability related to risks in source code and look for known patterns for common vulnerabilities and perform a line-by-line review of risky code to understand application logic and flaws that may exist [14, 16].

The code review phase could use tools, but it is necessary keep in mind that tools does not do all work. Manual review is always required.

Problems related with abuse cases and with risks specified above were found in Monitor software source code when performed a detailed code review. Field validation and authentication controls are not properly implemented. Examples of vulnerabilities found in source code review are shown in Table 1.

TABLE I. SOME PROBLEMS FOUND IN CODE REVIEW

#	Vulnerabilities		
	Flaws	*Class Name*	*Line Number*
1	Logging of user activity	Global (Many Classes)	N/A
2	No validation on input field	AccountMaintainActivity.java	142
3	Persistent Command Injection	UserDAO.java	119

To confirm that problems found in source code review could really be exploited, a penetration test must be performed. There are, at least, three phases involved in penetration testing: test preparation, test and test analysis as shown in Figure 6.

First phase is related to scope, objectives, timing and duration of the test. All legal agreements must be arranged during this phase. Second phase is considered the bulk of penetration test process. This phase involves application information gathering, vulnerability analysis and exploits. Results are investigated and analyzed in the last phase. The final report generated must be comprehensive and systematic [18].

Security operation is concerned with platform problems that could happen while software is working. Monitor software must be configured following Android Platform security specifications and requirements, as show in [13]. Examples of described requirements are data protection, cryptographic practicalities and use of protected communication channels.

This work is not confirming source code review with a penetration test since application is on early development stage. As soon as Monitor software development starts follow a security development plan, regular dynamic evaluation will be performed as soon as software becomes mature.

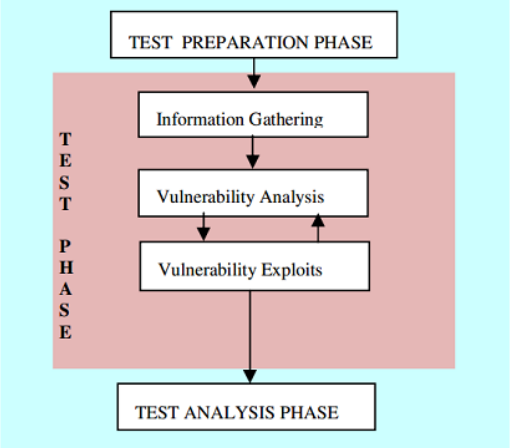

Figure 6. Penetration Test Phases [18].

Despite code review was not confirmed with penetration test, the common flaws shown in Table 1 are enough to demonstrate that touch points must be considered in software development lifecycle. A hacker or an attacker with moderated knowledge can exploit these software flaws easily.

VI. CONCLUSION AND FUTURE WORK

This assessment showed the importance of observing the security aspects in the software development lifecycle. The standards used for regulation of medical device software do not take into account security concerns. These aspects can make all difference in final software security and also in patient safety.

It is responsibility of *QSRs* deal with security concerns clearly. In general, standards for normalization of validations and verification are worried about functional aspects of software operation. Security issues are generally collateral problems that persist in all phases of software lifecycle, until software finishes its production life.

The monitor software used in the analysis was not designed, and as consequence, built with security concerns. So, every kind of security issue can appear in assessment. Since assessed software is in earlier stage of development, it is easier to map problems, flaws, issues and vulnerabilities and create a plan to mitigate them.

Generally, this kind of assessment produces lots of confidential results, and it is difficult to show them without brake non-disclosure agreements and/or reveal sensitive information about software internal structure. More relevant results were discussed directly with design and implementations teams involved in research project.

 ISBN: 978-1-61208-250-9

Unfortunately, securities problems are only solved when entire team involved in software construction are conscious about how it can affect in software operation.

To create this kind of conscience lots of actions are important. But, only organizations that have a security culture and security personal with secure coding and assessment skills can address these actions correctly.

In next steps, a complete penetration test will be performed, trying to exploit vulnerabilities found in code reviews and confirming that risks mapped were mitigated.

ACKNOWLEDGMENT

The authors would like to thank Software Engineer Group from Computer Science Department at University of Brasilia for allowing the analysis performed at Monitor software. This software is part of a research project entitled *Ambient Assisted Living Product Lines*.

REFERENCES

[1] H. Fraser, Y. Kwon, and M. Neuer, The future of connected health devices, IBM Institute for Business Value, New York, 2011.

[2] IEC 62304, Medical device software – Software life cycle processes, 1st ed, Geneva, 2006.

[3] ISO 27799, Health informatics – Information security management in health using ISO/IEC 27002, 1st ed, Geneva, 2008.

[4] J. Radcliffe, "Hacking Medical Devices for Fun and Insulin: Breaking the Human SCADA System", Black Hat Conference, Las Vegas, 2011, http://media.blackhat.com/bh-us-11/Radcliffe/BH_US_11_Radcliffe_Hacking_Medical_Devices_WP.pdf, 01.01.2013

[5] D. Halperin, et al., "Pacemakers and Implantable Cardiac Defibrillators: Software Radio Attacks and Zero-Power Defenses", Proceedings of the IEEE Symposium on Security and Privacy, Oakland, 2008, pp. 129-142, doi:10.1109/SP.2008.31.

[6] S. R. Rakitin, "Coping with Defective Software in Medical Devices", Computer Magazine - IEEE Computer Society, v. 39, 2006, n. 4, pp. 40-45, doi: 10.1109/MC.2006.123.

[7] D. A. Vogel, Medical Device Software Verification, Validation, and Compliance, Boston: Artech House, 2011.

[8] G. McGraw, Software security: building security in, Boston: Addison Wesley Professional, 2006.

[9] U.S. Food and Drug Administration, Guidance for the Content of Premarket Submissions for Software Contained in Medical Devices. 1st ed, New Hampshire, FDA, 2005.

[10] Open Web Application Security Project, OWASP Top Ten – 2010 The Ten Most Critical Web Application Security Risks, CC:OWASP, 2010.

[11] K. Hall, "Developing Medical Device Software to IEC 62304", European Medical Device Technology Magazine, v. 1, n. 6, June 2010.

[12] R. J. Anderson, Security Engineering: A Guide to Building Dependable Distributed Systems, 2nd ed, Indianapolis: Wiley Publishing inc, 2008.

[13] J. Six, Application Security for the Android Platform, 1st ed California: O'Reilly Media, Inc, 2012.

[14] F. Long, D. Mohindra, R. C. Seacord, D. F. Sutherland, and D. Svoboda, The CERT Oracle Secure Coding Standard for Java, 1st ed, Michigan: Pearson Education Inc, 2012.

[15] D. Stuttard and M. Pinto, The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws. 2nd ed Indianapolis: Wiley Publishing inc, 2011.

[16] M. Paul, Official (ISC)2 Guide to the CSSLP, 1st ed, Florida: CRC Press, 2011.

[17] V. Nunes, P. Fernandes, V. Alves, and G. Rodrigues, "Variability Management of Reliability Models in Software Product Lines: an Expressiveness and Scalability Analysis", I: SBCARS - Simpósio Brasileiro de Componentes, Arquitetura e Reutilização de Software, Natal - Brazil, 2012, pp. 113 - 122.

[18] A. G. Bacudio, X. Yuan, B. B. Chu, and M. Jones , "An Overview of Penetration Testing", International Journal of Network Security & Its Applications (IJNSA), v.3, 2011, n.6, pp. 19-38, doi: 10.5121/ijnsa.2011.3602.

 ISBN: 978-1-61208-250-9