UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO

SOFTWARE PARA VERIFICAÇÃO DE CONFORMIDADE DE SISTEMAS À NORMA ISO/IEC 15408

DAYANA FERNANDA TRAPP

BLUMENAU
2010

2010/1-07

DAYANA FERNANDA TRAPP

SOFTWARE PARA VERIFICAÇÃO DE CONFORMIDADE DE SISTEMAS À NORMA ISO/IEC 15408

Trabalho de Conclusão de Curso submetido à Universidade Regional de Blumenau para a obtenção dos créditos na disciplina Trabalho de Conclusão de Curso II do curso de Ciência da Computação — Bacharelado.

Prof.. Paulo Fernando da Silva - Orientador

BLUMENAU
2010

2010/2-07

SOFTWARE PARA VERIFICAÇÃO DE CONFORMIDADE DE SISTEMAS À NORMA ISO/IEC 15408

Por

DAYANA FERNANDA TRAPP

Trabalho aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II, pela banca examinadora formada por:

Presidente:	Prof. Paulo Fernando da Silva, Mestre – Orientador, FURB
Membro:	Prof. Everaldo Artur Grahl, Mestre – FURB
Membro:	Prof. Marcel Hugo, Mestre – FURB

Blumenau, 05 de julho de 2010

Dedico este trabalho aos meus pais que sempre me incentivaram a estudar.

AGRADECIMENTOS

A Deus, pela família que me deu, e pela força para o término deste trabalho.

Aos meus pais que sempre estiveram ao meu lado, cuidando, dando amor, carinho, apoio aos estudos e dando exemplos a serem seguidos.

Aos meus amigos que fiz durante essa vida que ainda estão comigo. A família que consisti durante os anos de faculdade, principalmente ao Ivan que sempre esteve ao meu lado nestes últimos anos de faculdade, sempre pronto para me ajudar nas dúvidas da faculdade, em me escutar quando precisava, de trocar idéias, músicas, compreensão e por ser um dos responsáveis pela pessoa que sou hoje.

A Senior Sistemas pelo entendimento das ausências, e os colaboradores que me apoiaram e trocaram idéias.

Ao meu orientador, Paulo Fernando da Silva que me orientou, incentivou para que eu concluísse o trabalho.

E por fim, a este trabalho, pelo conhecimento, desafio e superação, apesar do estresse.

Pequenos atos ou pequenos grandes momentos podem fazer a diferença.

Dady

RESUMO

Este trabalho apresenta a especificação de um software para auditar o desenvolvimento de sistemas, de acordo com a norma de segurança ISO/IEC 15408. O software é voltado para auditores de segurança e contém como principal funcionalidade a realização de testes automatizados para cada requisito da norma. Os testes são acoplados no software através de *plugins*, que o próprio auditor pode desenvolver utilizando a API disponibilizada pelo software.

Palavras-chave: Segurança. Software. Auditoria. ISO/IEC 15408.

ABSTRACT

This paper presents the specification of a software to audit the systems development, according to the safety standard ISO / IEC 15408. It is aimed for security auditors and contains as its main functionality for automated testing each requirement of the standard. Tests are engaged in software through plugins, which the auditor may develop using the API provided by the software.

Key-words: Security. Software. Auditing. ISO/IEC 15408.

LISTA DE ILUSTRAÇÕES

Figura 1 - Priorização de ameaças 18
Figura 2 - Melhorias incrementais da segurança no processo de desenvolvimento 21
Figura 3 – Evolução do critério de segurança 24
Figura 4 - Software para verificação de conformidade de servidores GNU/Linux à norma de segurança NBR ISO/IEC 27002 27
Figura 5 - Software para avaliação da segurança da informação de uma empresa conforme a Norma NBR ISO/IEC 17799 28
Figura 6 – Caso de uso criação do plano de auditoria 31
Quadro 1 - Detalhamento do caso de uso `Cadastra dados da auditoria` 32
Quadro 2 – Detalhamento do caso de uso `Seleciona requisitos a serem avaliados` 32
Figura 7 - Caso de uso realização da auditoria 32
Quadro 3 – Caso de uso `Realiza auditoria automática` 33
Quadro 4 – Caso de uso `Gera relatório` 33
Figura 9 – Diagrama de classe correspondente ao pacote `Software` 35
Figura 10 – Diagrama de classe correspondente ao pacote `API` 37
Figura 11 – Diagrama de classe correspondente ao pacote `Plugin` 38
Figura 12 – Diagrama de classe correspondente ao pacote `interface` 39
Figura 13 – Diagrama de sequência da abertura do software 40
Figura 14 – Diagrama de sequência para executar a auditoria automática 40
Quadro 5 – Estrutura do arquivo XML de armazenamento da árvore dos requisitos 41
Quadro 6 - Parte do código da classe `LoadXML` 42
Quadro 7 – Exemplo de código que procura um fonte Java com uma determinada interface 43
Quadro 9 – Parte do código que carrega as classes que implementam a interface `IAuditing` 44
Quadro 10 – Interface `IAuditing` 45
Quadro 11 – Rotina que itera pela lista de `packege` e executa a conformidade com a norma 47
Quadro 12 – Rotina que procura pelo `import` de `Logger` 47
Quadro 13 – Carregamento do classpath do sistema pelo *plugin* 48

Quadro 14 – Rotina de verificação da mensagem do *login* 49
Figura 15 - Criar um novo plano de auditoria 50
Figura 16 - Cadastro básico do sistema auditado 50
Figura 17 - Seleção dos requisitos a serem auditados 51
Figura 18 – Execução da auditoria 51
Figura 19 - Tela de execução dos requisitos da norma 52
Figura 20 - Resultados da auditoria 52
Figura 21 - Gerar relatório da auditoria 53
Figura 22 – Relatório com os dados da auditoria 53
Figura 23 – Relatório com os resultados da auditoria 54
Quadro 15 - Classe auditoria de segurança 60
Quadro 16 – Classe comunicação 60
Quadro 17 – Classe Proteção de dados do usuário 61
Quadro 18 – Identificação e autenticação 62
Quadro 19 – Classe Gerenciamento de segurança 63
Quadro 20 – Classe Autoproteção 64
Quadro 21 –Classe Utilização de recursos 65
Quadro 22 – Classe Acesso ao sistema 65
Quadro 23 – Classe Caminhos ou canais confiáveis 65

LISTA DE SIGLAS

API – *Application Programming Interface*

BCC – Bacharelado em Ciência da Computação

CC – *Common Criteria*

CTCPEC – *Canadian Trusted Computer Product Evaluation*

DSC – Departamento de Sistemas e Computação

EAL – *Evaluation Assurance Level*

FC – *Federal Criteria*

GWT – *Google Web Toolkit*

IDE – *Integrated Development Environment*

ISO – *International Standardization Organization*

ITSEC – *Information Technology Security Evaluation Criteria*

PP – *Protection Profile*

ST – *Security Target*

TCSEC – *Trusted Computer Security Evaluation Criteria*

TI – Tecnologia da Informação

TOE – *Target Of Evaluation*

UC – Caso de uso

UML – *Unified Modeling Language*

XML – *eXtensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO 13
1.1 OBJETIVOS DO TRABALHO 14
1.2 ESTRUTURA DO TRABALHO 15
2 FUNDAMENTAÇÃO TEÓRICA 16
2.1 SEGURANÇA DA INFORMAÇÃO 16
2.2 DESENVOLVIMENTO DE SOFTWARE SEGURO 18
2.2.1 Desenvolvimento da segurança nas empresas 19
2.2.2 Processos de desenvolvimento 20
2.2.3 Boas práticas de programação 22
2.2.4 Implementação de segurança no JAVA 23
2.3 NORMA ISO/IEC 15408 23
2.4 AUDITORIA DE SOFTWARE 26
2.5 TRABALHOS CORRELATOS 26
2.5.1 Software para verificação de conformidade de servidores GNU/Linux à norma de segurança NBR ISO/IEC 27002 27
2.5.2 Software para avaliação da segurança da informação de uma empresa conforme a Norma NBR ISO/IEC 17799 28
2.5.3 Fortify SCA 29
3 DESENVOLVIMENTO 30
3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO 30
3.2 ESPECIFICAÇÃO 31
3.2.1 Diagrama de Casos de Uso 31
3.2.2 Arquitetura do Software 33
3.2.3 Diagrama de classe 34
3.2.4 Diagrama de seqüência 39
3.3 IMPLEMENTAÇÃO 41
3.3.1 Técnicas e ferramentas utilizadas 41
3.3.2 Dom4j 41
3.3.3 QDox 42
3.3.4 iText 43
3.3.5 Desenvolvimento do plano de auditoria 44

3.3.6 Desenvolvimento da API para *plugin* 45
3.3.7 Exemplo de teste automatizado 46
3.3.7.1 Teste FAU_GEN.1 46
3.3.7.2 Teste FAU_GEN.2 47
3.3.7.3 Teste FIA_UAU.7 49
3.3.8 Operacionalidade da implementação 50
3.4 RESULTADOS E DISCUSSÃO 54
4 CONCLUSÕES 56
4.1 EXTENSÕES 57
REFERÊNCIAS BIBLIOGRÁFICAS 58
ANEXO A – Divisão dos requisitos funcionais da norma 15.408 60

1 INTRODUÇÃO

Para Albuquerque e Ribeiro (2002, p. 1), o desenvolvimento de sistemas é uma das áreas mais afetadas pelos aspectos da segurança. Muitos dos problemas de segurança existentes hoje, não são, nem físicos e nem de procedimento, mas sim, devidos a erros de programação ou de arquitetura.

Segundo Guerra (2008, p. 26), para não perder muito tempo e entregar a solução o quanto antes para o cliente, os requisitos de segurança são deixados de lado. Os clientes por sua vez não possuem noção sobre segurança de um sistema e só saberão mais tarde quando é encontrada uma vulnerabilidade. Isso acontece porque poucos analistas preocupam-se em especificar bem os requisitos de segurança.

De acordo com Albuquerque e Ribeiro (2002, p. 1), a segurança em sistemas sempre foi importante e com a internet a segurança torna-se o foco, uma vez que os sistemas tendem a ficar mais interconectados, facilitando o acesso do *cracker*[1]. Assim, a segurança será cada vez mais uma preocupação no desenvolvimento de sistemas.

Tem-se como premissa que nenhum software é seguro (OLIVEIRA, 2001, p. 11). A segurança de um software é afetada porque ele pode executar outros procedimentos os quais não foram propostos. Se ele fizesse exatamente o que foi destinado a fazer, a segurança não seria uma preocupação (CONALLEN, 2003, p. 76). Portanto, existe uma facilidade para invasores investigarem vulnerabilidades desconhecidas e dificuldade dos desenvolvedores em garantir que todos os pontos de entrada do sistema estejam protegidos (HOWARD; LEBLANC, 2005, p. 48).

A segurança deve ser uma preocupação corporativa. Os usuários exigem que os softwares que eles utilizam sejam seguros, isso é um direito deles, e não um privilégio. Porém, a não preocupação em desenvolver segurança nos sistemas leva, no final das contas, a um maior volume de trabalho e a uma reputação ruim, já que pode chegar a perda de vendas, à medida que os clientes troquem o produto pelo do concorrente que é mais seguro (HOWARD; LEBLANC, 2005, p. 36-37).

Para saber se o sistema é seguro, na década de 80 surgiu o primeiro padrão para avaliação de segurança em softwares, que ficou conhecido como *Orange Book*. Mais tarde este padrão foi homologado pela *International Standartization Organization* (ISO) como

[1] *Cracker* é uma pessoa que viola maliciosamente a segurança de um sistema.

ISO/IEC 15408, que muitas vezes é chamada apenas de *Common Criteria* (CC) (AZEVEDO, 2008). De acordo com Albuquerque e Ribeiro (2002, p. 7), a norma ISO/IEC 15408 é o melhor ponto de partida para o desenvolvimento de software seguro, pois ela descreve conceitos necessários para a segurança em sistemas.

O CC determina que um sistema deva ter seu *Security Target* (ST) (objetivo ou alvo de segurança) definido, para ser considerado seguro. "O ST é a especificação de segurança, ou seja, indica quais aspectos de segurança foram considerados importantes e porque o foram para aquele sistema em particular" (LYRA, 2008, p. 71).

Desta forma, para auxiliar um auditor na avaliação de um sistema, propõe-se o desenvolvimento de um software que fará a análise do sistema, e identificará automaticamente quais requisitos da norma são atendidos. Porém, nem todos os requisitos estarão disponíveis para a análise automática. Então, para estes requisitos o software fornecerá uma lista com os requisitos e campos de observações para que o auditor faça a avaliação de modo manual, entrando com os dados da avaliação realizada, para que no final seja gerado um relatório com o resultado da auditoria. Para não gerar ambigüidade utiliza-se a palavra software para identificar o software desenvolvido neste trabalho e sistema para o sistema que sofrerá a auditoria.

1.1 OBJETIVOS DO TRABALHO

O objetivo deste trabalho foi desenvolver um software para analisar um sistema e verificar se o mesmo implementa requisitos de segurança estabelecidos pela norma ISO/IEC 15408. Os objetivos específicos do trabalho:

a) analisar um sistema e verificar se ele implementa requisitos da norma ISO/IEC 15408, tais como auditoria de segurança, proteção de dados do usuário e autentic,ação;
b) disponibilizar uma API para que possa ser implementado os testes automatizados e acoplado no software através de *plugin*;
c) disponibilizar em forma de *check list* os requisitos que não serão analisados automaticamente para que o auditor faça a auditoria manual;
d) disponibilizar um relatório do sistema avaliado, assinalando os requisitos atendidos.

1.2 ESTRUTURA DO TRABALHO

O trabalho está organizado em três capítulos, intitulados respectivamente como fundamentação teórica, desenvolvimento e conclusões.

O capítulo 2 apresenta os aspectos teóricos estudados para o desenvolvimento do trabalho. São abordados os seguintes temas: segurança da informação, desenvolvimento de software seguro, norma ISO/IEC 15408, auditoria de software e por fim são apresentados os trabalhos correlatos.

O capítulo três é apresentado a especificação do trabalho, principais caso de uso, bem como os requisitos do sistema.

Por último, no capítulo 4 tem-se a conclusão do trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta inicialmente os conceitos de segurança. Em seguida aborda a importância de se desenvolver um software seguro. Posteriormente é apresentada a norma de segurança ISO/IEC 15408 para o desenvolvimento de software. Logo após são descritos os conceitos de uma auditoria de software. Por fim, são apresentados os trabalhos correlatos.

2.1 SEGURANÇA DA INFORMAÇÃO

De acordo com Dias (2000, p. 41) a segurança da informação tem como objetivo a proteção da informação, para reduzir a probabilidade e o impacto de incidentes de segurança. Segundo Lyra (2008, p. 4) um incidente de segurança ocorre quando um evento pode causar interrupções nos processos de negócio em decorrência da violação de alguma propriedade de segurança. Lyra (2008, p. 3) conceitua do seguinte modo as três propriedades básicas da segurança:

a) confidencialidade: capacidade de um sistema permitir que alguns usuários acessem determinadas informações ao mesmo tempo e impedir que outros, não autorizados, a vejam;
b) integridade: a informação deve estar correta, ser verdadeira e não estar corrompida;
c) disponibilidade: a informação deve estar disponível para todos que precisarem dela para a realização dos objetivos empresariais.

Além destas três propriedades principais, têm-se as seguintes propriedades conceituadas por Azevedo (2008):

a) autenticação: capacidade de garantir que um usuário, sistema ou informação é mesmo quem alega ser;
b) não repúdio: capacidade do sistema de provar que um usuário executou determinada ação no sistema;
c) legalidade: aderência de um sistema a legislação;
d) privacidade: capacidade que um sistema tem em manter incógnito um usuário, impossibilitado a ligação direta da identidade do usuário com as ações por este

realizadas;

e) auditoria: capacidade do sistema em verificar tudo o que foi realizado pelos usuários, detectando fraudes ou tentativas de ataque. Note-se que, é um aspecto impossível de se conciliar totalmente com a privacidade.

Quando se tem a perda de qualquer propriedade da segurança importante para o sistema, tem-se um problema de segurança. Assim, trabalha-se na prevenção para manter as propriedades de segurança dentro das necessidades do cliente e evitar falhas (ALBURQUERQUE; RIBEIRO, 2002, p. 3). Por exemplo, como relata Dias (2002, p. 44) um sistema que deve estar disponível 24 horas, deve ter alta disponibilidade e não necessita tanto tratar confidencialidade e privacidade dos dados. Nos sistemas militares ou de segurança nacional, o ponto critico é a privacidade, devido as informações confidenciais. Já em sistemas bancários, a integridade e a auditoria são os pontos mais relevantes. Para cada tipo de sistema a necessidade de segurança é diferente, por isso, deve ser tratada de forma distinta para cada caso.

Azevedo (2008) observa que as falhas de segurança em software, são as grandes responsáveis pelos ataques em sistemas. Allen et al. (2001) coloca que a exposição que vem com a conectividade global torna a informação sensível e o sistema mais vulnerável para o uso não autorizado e intencional. O crescimento da conectividade da internet pelos computadores e a dependência dos usuários de serviços on-line tem feito grandes números de ataques sofisticados.

Para Alburquerque e Ribeiro (2002 p. 3), o ataque ao sistema é um problema de segurança grave, pois o agente que está realizando o ataque visa obter alguma vantagem, podendo com isso, provocar grandes prejuízos. Há sempre três elementos no ataque: o agente que realiza o ataque, um ativo com valor, e uma vulnerabilidade, ou seja, uma forma de atingir o ativo. Segue as seguintes definições:

a) ataque: é um tipo de problema de segurança caracterizado pela existência de um agente que busca obter algum tipo de retorno, atingindo um ativo de valor, por exemplo, dinheiro;

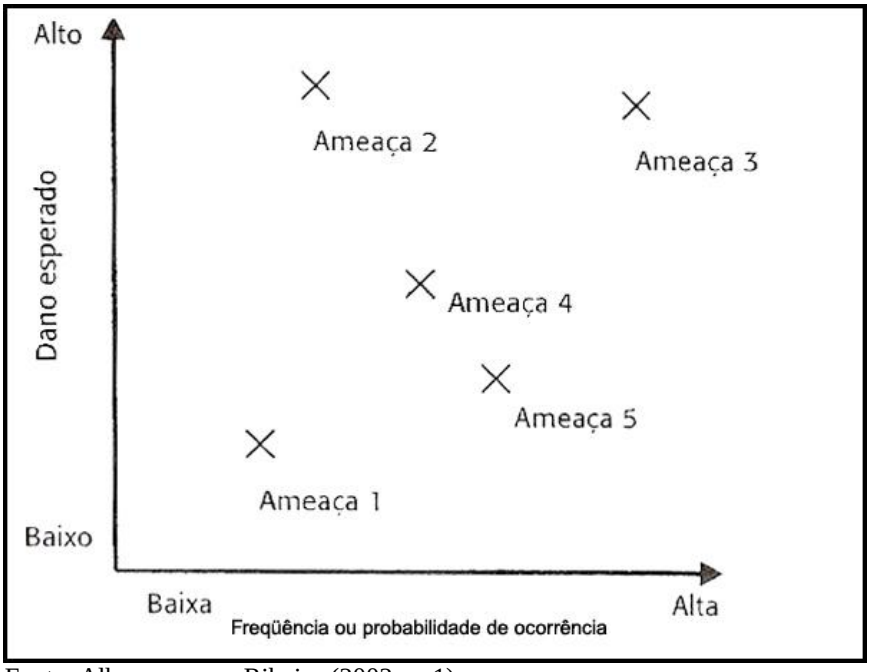
b) ativo: algo de valor protegido pelo sistema. Geralmente está aliado com uma propriedade de segurança, por exemplo, a confidencialidade dos dados do orçamento;

c) vulnerabilidade: uma falha no sistema que é explorada em todo o ataque. O ataque pode ser intencional ou pode ocorrer por erros de operação e que permita que o ativo seja atingido;

d) ameaça: é um ataque a um ativo da informação. É um agente externo que, aproveitando-se da vulnerabilidade, poderá quebrar um ou mais dos três princípios de segurança.

Brandão e Fraga (2008) afirmam que o sistema está em risco quando apresenta vulnerabilidades que podem ser exploradas produzindo algum impacto negativo. Lyra (2008, p. 6) observa que pode se ter um ativo com várias vulnerabilidades, mas sem ameaças de ataque, o que se leva a ter uma probabilidade próxima de 0. A probabilidade é a chance de uma falha de segurança ocorrer, levando-se em conta as vulnerabilidades do ativo e as ameaças que venham a explorar esta vulnerabilidade

A Figura 1 mostra a probabilidade de ocorrência dos ataques e qual é o dano esperado, ou seja, deve-se priorizar a correção da ameaça 3.



Fonte: Albuquerque e Ribeiro (2002, p. 1).

Figura 1 - Priorização de ameaças

Howard e Leblanc (2005, p. 48) concluem que o defensor pode se defender somente de ataques conhecidos, já o invasor pode investigar as vulnerabilidades desconhecidas.

2.2 DESENVOLVIMENTO DE SOFTWARE SEGURO

Oliveira (2001, p. 49) relata que as tecnologias utilizadas em larga escala na internet para transações comercias, não suportam os diversos requisitos relativos a segurança como

um todo. A autenticidade, a integridade, o sigilo e a aceitação, consistem nos principais fundamentos para que os objetivos de segurança possam ser inteiramente aplicados. Deste modo, Albuquerque e Ribeiro (2002, p. 6) afirmam que a segurança será cada vez mais uma preocupação central no desenvolvimento de qualquer sistema e que se deve preocupar com as seguintes questões:

a) segurança do ambiente de desenvolvimento: ter o comprometimento da equipe e evitar roubo do código-fonte;
b) segurança da aplicação desenvolvida: seguir a especificação de segurança, evitar acessos ocultos (*backdoor*) e falhas que comprometam a segurança;
c) garantia de segurança da aplicação desenvolvida: garantir para o cliente que a aplicação em desenvolvimento é segura.

Essas questões estão sempre interligadas uma com a outra, não há como se conseguir uma aplicação segura, sem um ambiente de desenvolvimento seguro.

2.2.1 Desenvolvimento da segurança nas empresas

Conforme Conallen (2003, p. 77) o maior responsável por ataques são as falhas de segurança contidas nos softwares. É tarefa dos arquitetos entenderem e administrarem as falhas, já que possuem um bom conhecimento do sistema.

Algumas empresas não se preocupam com os profissionais que irão desenvolver os sistemas colocando, por exemplo, funcionários não especializados em programação, como *web designers*, para programar. Pela falta de conhecimento, acabam gerando falhas no código e deixando o sistema vulnerável (THOMPSON, 2002, p. 17).

Para Dias (2000, p. 44) as empresas deixam a segurança de lado, e só se lembram depois que o desastre tenha ocorrido. Howard e Leblanc (2005, p. 37) listam algumas razões para as pessoas não se preocuparem com a segurança:

a) a segurança é entediante;
b) a segurança costuma ser vista como um desativador de funcionalidade, como algo que atrapalha;
c) a segurança é difícil de medir;
d) normalmente, a segurança não é a principal habilidade ou interesse dos projetistas e desenvolvedores que criam o produto;
e) a segurança não significa criar algo novo e animador.

Howard e Leblanc (2005, p. 34-35) afirmam que sistemas seguros possuem qualidade. Um código projetado e construído com segurança como recurso principal é mais robusto do que um código em que a segurança é pensada posteriormente.

2.2.2 Processos de desenvolvimento

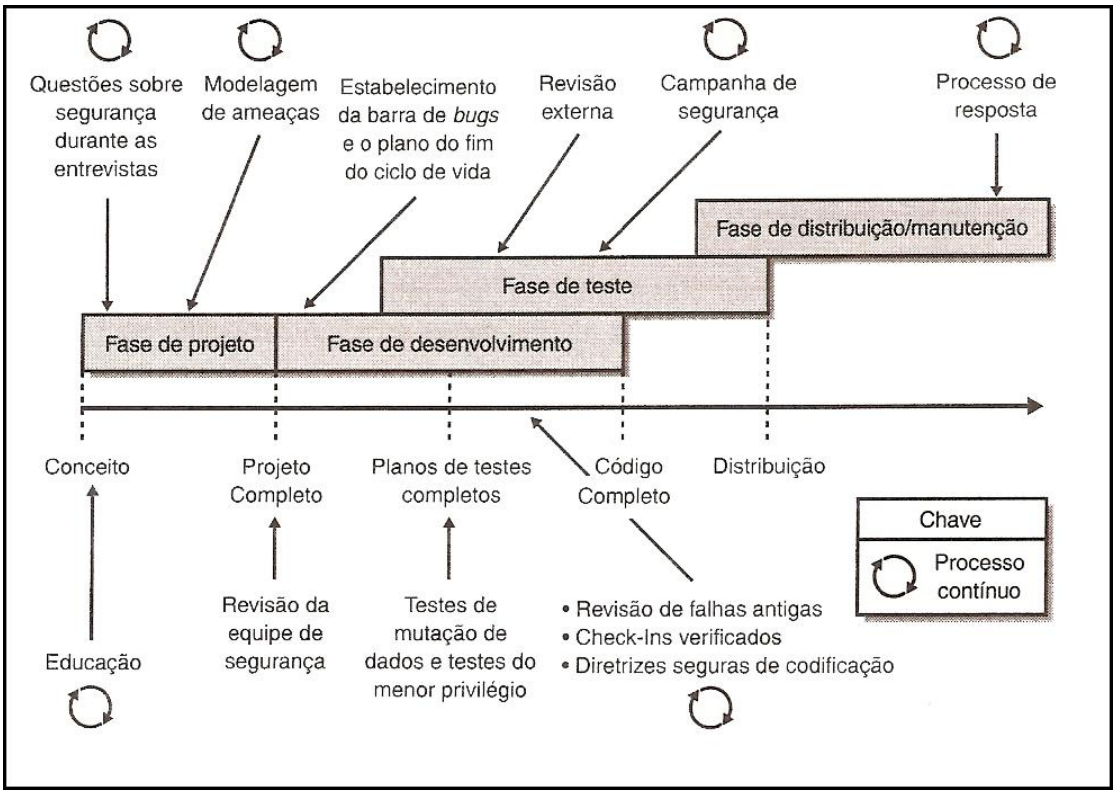
Assim como na vida fora dos *bits* e *bytes* o risco sempre vai existir. O dilema é trazer o "grau de risco" a um nível aceitável, de modo que se pode evoluir na utilização da tecnologia até um patamar mais confiável e eficaz (OLIVEIRA, 20 p. 14).

De acordo com Oliveira (2001, p. 11), é impossível ter um sistema totalmente seguro. Brandão e Fraga (2008) defendem que os riscos podem ser amenizados quando são identificados, mas nunca totalmente eliminados. O sistema seguro é aquele que concentra suas defesas nos ataques mais prováveis e com maior perda esperada (ALBUQUERQUE; RIBEIRO, 2002, p. 4).

Para Lyra (2008, p. 71) as funcionalidades de segurança devem ser representadas nos vários níveis de abstração, desde o projeto lógico, até a implementação de seus produtos finais. Assim, Nunes e Belchior (2006) afirmam que se uma empresa deseja começar a implementar segurança, ela deverá primeiro fazer o planejamento de segurança, que forneça as seguintes atividades demonstradas:

a) definir planejamento de segurança e identificar seus mecanismos;
b) atribuir responsabilidades de segurança no projeto;
c) implementar ambientes de processamento;
d) planejar o gerenciamento de incidentes de segurança.

Para as empresas que seguem algum tipo de processo de desenvolvimento de software, Howard e Leblanc (2005, p. 52-53) sugerem a atualização do processo, incluindo aprimoramentos de segurança em cada etapa do ciclo de vida. Na Figura 2 é exibido um modelo em cascata acrescentando as questões de segurança em cada parte do processo.



Fonte: Howard e Leblanc (2005, p. 34-35).

Figura 2 - Melhorias incrementais da segurança no processo de desenvolvimento

Howard e Leblanc (2005, p. 55-53) afirmam que a preocupação já deve vir na contratação do profissional que irá trabalhar no projeto, verificar se ele possui habilidades de segurança. Treinar os profissionais em segurança também é uma boa prática.

De acordo com Albuquerque e Ribeiro (2002, p. 11), existem ameaças ao negócio ou ao objeto da aplicação, que precisam ser eliminadas ou, ao menos, mitigadas. Howard e Leblanc (2005, p. 55-53) afirmam que é necessário determinar quem é o público-alvo e quais serão os requisitos de segurança que o sistema conterá e coloca as seguintes questões:

a) quem é o público do aplicativo?
b) o que a segurança significa para o público? Há alguma distinção entre diferentes membros do público? Os requisitos de segurança são diferentes para diferentes clientes?
c) onde o aplicativo será executado? Na internet? Por trás de um *firewall*? Em um telefone celular?
d) o que você está tentando proteger?
e) quais são as implicações para os usuários, se os objetos que você está protegendo forem comprometidos?
f) quem vai gerenciar o aplicativo? O usuário ou um administrador de Tecnologia da Informação (TI) corporativo?

g) quais são as necessidades de comunicação do produto? O produto é interno ou externo à organização, ou ambos?
h) quais serviços da infra-estrutura de segurança que o sistema operacional e o ambiente já fornecem e de que você pode tirar proveito?
i) como o usuário precisa ser protegido de suas próprias ações?

Segundo Lyra (2008, p. 75) tendo feito o levantamento de ameaças tem-se o objetivo de segurança, cada ameaça está relacionada com um objetivo de segurança. Porém deve-se cuidar para não gerar um plano muito caro, corrigindo todas as ameaças, mas sim, tratar somente as ameaças significativas, ou com maior probabilidade de impacto. Proteger o sistema das principais ameaças dificulta os ataques.

Desta forma, são levantadas as métricas para o desenvolvimento de software, medição de riscos, e quanto a empresa deverá investir em segurança. Levantando as métricas, pode-se verificar qual o nível de segurança que o sistema deve atingir (BATISTA, 2007 p. 38).

2.2.3 Boas práticas de programação

Para Lyra (2008, p. 86) os padrões de desenvolvimento visam melhorar a estrutura interna de um programa, garantir maior legibilidade do código, padronização da codificação, facilidade na manutenção, maior produtividade, rapidez no desenvolvimento, redução da quantidade de códigos e facilidade da detecção de erros.

Albuquerque e Ribeiro (2002, p. 6) afirmam que seguindo as normas da boa programação, automaticamente são eliminados muitos dos erros e falhas de segurança em uma aplicação, destacam-se os seguintes:

a) funções intrinsecamente seguras: tratar todas as variáveis de entrada como não-confiáveis, verificando sua validade antes de usá-las;
b) sempre verificar os códigos de erro retornados por uma função, principalmente nas chamadas a *Application Programming Interface* (API) do sistema operacional;
c) atentar sempre para o tamanho dos *buffers* e *arrays* do sistema;
d) documentar o código: se for trabalho em equipe, é indispensável para que não ocorra falha de comunicação entre a equipe.

Segundo Albuquerque e Ribeiro (2002, p. 11) além das boas práticas de programação existem legislações ou políticas de segurança que é preciso obedecer. Para isso foi criada a norma ISO/IEC 15408, que dita um processo de desenvolvimento de software seguro, com a

realização de testes e acompanhamento do projeto. Ela também fornece base para certificar os produtos, avaliando a sua segurança. A segurança de um determinado sistema está nas métricas levantadas, que serão verificadas e classificadas em uma determinada classe de produto (BATISTA, 2007 p. 39).

2.2.4 Implementação de segurança no JAVA

Para o desenvolvimento de autenticação no Java, em aplicativos voltados para web encontram-se algumas possíveis implementações (IMASTERS, 2009):

a) Servlet Filter: maneira mais simples de implementar controle de usuários. Ela é feita através da construção e configuração manual de *servlet*;
b) Phase Listener: é utilizado por aplicações desenvolvidas em Java Server Faces (JSF). Possui um recurso chamado *listener* que é responsável por interceptar e oferecer mecanismos de manipulações referentes a mudanças de eventos ocorridas no ciclo de vida da aplicação, sendo assim útil para o controle de usuários;
c) Java Authentication and Authorization Service: Oferece interfaces para definir e gerenciar controle de acesso em aplicações JEE.
d) jGuard: é um framework baseado em JAAS, oferecendo facilidades e integrações com alguns componentes JEE;
e) Spring Security: é um framework voltado ao desenvolvimento JAVA, possui recursos próprios para a implementação de autenticação e autorização.

2.3 NORMA ISO/IEC 15408

A norma ISO/IEC 15408 surgiu para unificar padrões de segurança e eliminar as diferenças de critérios. O primeiro padrão para avaliação da segurança surgiu nos EUA na década 1980. Foi nomeado de *Trusted Computer System Evaluation Criteria* (TCSEC) mais conhecido como *Orange Book* devido a sua capa laranja (ALBUQUERQUE; RIBEIRO, 2002, p. 7).

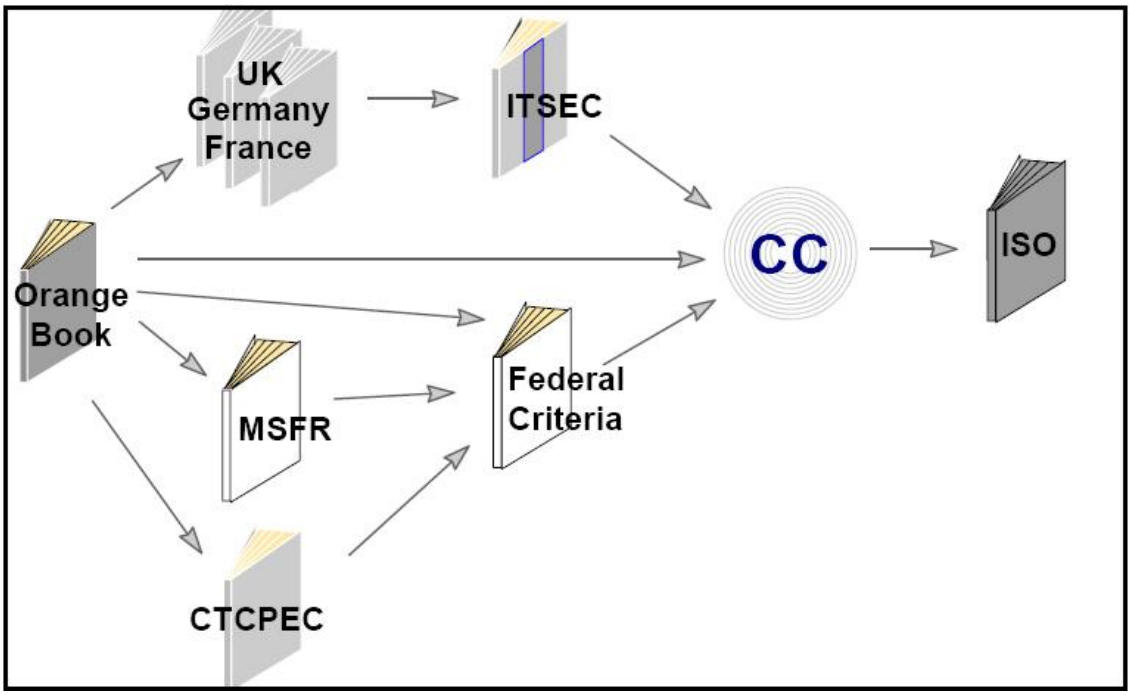
Em 1991 foi criada a norma para certificação de segurança chamada de *Information Technology Security Evaluation Criteria* (ITSEC) por uma comissão européia envolvendo os

países Alemanha, França. Holanda e Reino Unido. O ITSEC se tornou um padrão europeu voltado tanto para a avaliação de produtos, como de sistemas (BATISTA, 2007, p. 42).

Albuquerque e Ribeiro (2002, p. 8) relatam que outros padrões surgiram na década de 1990, como *Canadian Trusted Computer Product Evaluation* (CTCPEC) que é a junção das normas TCSEC e ITSEC e o *Federal Criteria* (FC) desenvolvida pelos EUA.

Com o grande número de normas, as empresas internacionais começaram a ter problemas para certificarem seus produtos. Assim foi sugerida à ISO a criação de um critério único de avaliação de segurança, surgindo assim a norma ISO 15.408:1999, dando-lhe o nome de *Common Criteria* (CC) (BATISTA, 2007, p. 43).

Na Figura 3 é representada a evolução do critério de segurança mencionado acima.



Fonte: Oracle (2001).

Figura 3 – Evolução do critério de segurança

O CC é um padrão internacional de desenvolvimento de produtos seguros, o qual descreve uma lista de critérios de segurança que um produto deve ter. Assim, a norma desenvolve um critério de avaliação de segurança da informação, dando segurança aos clientes, testando e certificando seus produtos e serviços (BURNETT; PAINE, 2002, p. 284).

O CC está dividido em três partes (COMMOM CRITERIA, 2009):

a) descreve a introdução do CC. Define o conceito e os princípios gerais da avaliação da segurança da informação em TI e apresenta a lista de requisitos de segurança, que estão divididos em classe, família e requisito;
b) cataloga uma série de requisitos funcionais e organizados em famílias e classe para

a avaliação do *Target Of Evaluation*[2] (TOE). Estes requisitos estão no Anexo A;

c) Define o critério de avaliação para o *Protection Profile* (PP) e *Security Target* (ST) se apresenta sete pacotes de segurança pré-definidos que são chamado de *Evaluation Assurance Level* (EAL).

O CC estabelece que qualquer sistema, para ser considerado seguro, deve ter seu ST elaborado. O ST é a especificação de segurança, ou seja, indica quais aspectos de segurança foram considerados importantes e por que o foram, para aquele sistema em particular. Existe também o PP, que é um documento semelhante ao ST, com a diferença que não especifica uma única aplicação, podendo ser aplicação a toda uma classe de sistema (ALBUQUERQUE; RIBEIRO, 2002, p. 8).

O CC define sete níveis para garantir segurança, denominados de EAL. Para cada nível, tem-se um maior número de testes e, portanto, uma maior garantia que o sistema atende aos requisitos de segurança (ALBUQUERQUE; RIBEIRO, 2002, p. 8).

Os níveis da norma ISO/IEC 15408 são (BRANDÃO; FRAGA, 2008):

a) EAL1: certifica que o TOE teve seu funcionamento testado;
b) EAL2: estabelece que o sistema teve sua estrutura testada. Envolve a cooperação do fabricante;
c) EAL3: certifica que o TOE foi metodicamente testado e checado;
d) EAL4: define que o sistema foi metodicamente projetado, testado e checado;
e) EAL5: prevê que o sistema seja implementado e testado de maneira não formal;
f) EAL6: certifica que o TOE foi projetado, verificado e testado de maneira não formal;
g) EAL7: define que o sistema foi projetado, verificado e testado de maneira formal.

É fácil ver que, atingir o nível EAL7 de segurança leva tempo e dinheiro. Para os sistemas comerciais, o nível EAL3 traz uma segurança significativa e com um custo menor (LYRA, 2008, p. 72).

Alburquerque e Ribeiro (2002, p. 9) defendem que seguir os padrões e a ideia do CC já se pode obter um nível elevado de segurança no desenvolvimento dos sistemas, pois aplicar a norma em sua totalidade exige muitos detalhes, avaliações por laboratórios internacionais e o custo é alto.

---

[2] *Target of Evaluation* é o sistema que está sendo avaliado

2.4 AUDITORIA DE SOFTWARE

> A auditoria é uma atividade que engloba o exame das operações, processos sistemas e responsabilidades gerenciais de uma determinada entidade, com intuito de verificar sua conformidade com certos objetivos e políticas institucionais, orçamentos, regras, normas ou padrões (DIAS, 2000, p. 8).

Dias (2000, p. 13-14) relata a existência de três modelos de auditoria da segurança:

a) auditoria da segurança de informações: determina a postura da organização com relação a segurança, que geralmente envolve itens como controle de acesso lógico, físicos e ambientais;
b) auditoria da tecnologia da informação: abrange toda a segurança da informação, como também os controles organizacionais, de operação de sistemas e sobre o banco de dados;
c) auditoria de aplicativos: determina a segurança em aplicativos específicos tendo, como controles o desenvolvimento de sistemas aplicativos, de entrada processamento de entrada e saída de dados, sobre o conteúdo e funcionamento do aplicativo.

Para ser um auditor na área da tecnologia de informação, deve-se ter um bom conhecimento na área de sistemas computacionais, de preferência já ter trabalhado nela. O nível de conhecimento varia de acordo com os requisitos que o auditor irá avaliar (DIAS, 2000, p. 14).

Segundo Lyra (2008, p. 111) primeiro inicia-se o processo de levantamento das informações relevantes sobre o sistema. Esse levantamento deve ser feito de modo abrangente, de forma que seja possível o entendimento macro das características do sistema.

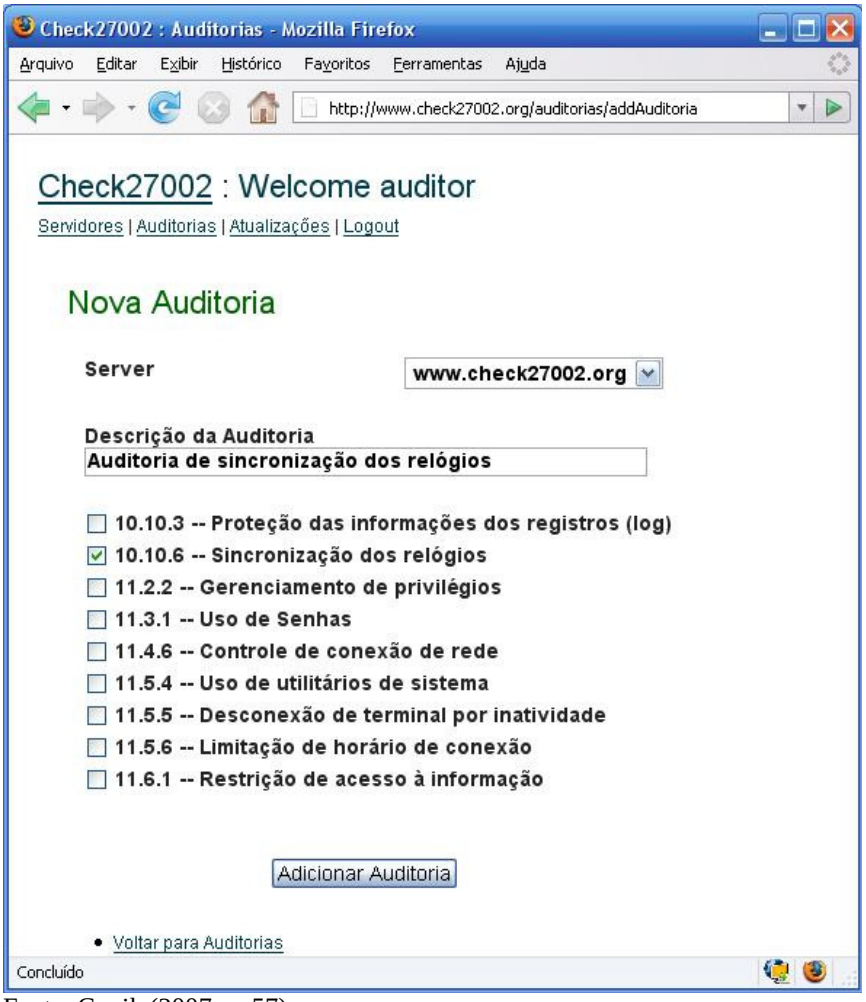
2.5 TRABALHOS CORRELATOS

Existem sistemas, incluindo ferramentas comerciais, semelhantes ao trabalho proposto. Dentre eles, foram escolhidos três cujas características enquadram-se nos principais objetivos deste trabalho, os quais são: “Software para Verificação de Conformidade de Servidores Gnu/Linux à Norma de Segurança NBR ISO/IEC 27002” (CUGIK, 2007), “Software para Avaliação da Segurança da Informação de uma Empresa Conforme a Norma NBR ISO/IEC

17799" (ROSEMANN, 2002) e a ferramenta de análise de código "*Fortify* SCA" (BRAZ, 2008).

2.5.1 Software para verificação de conformidade de servidores GNU/Linux à norma de segurança NBR ISO/IEC 27002

É um software livre desenvolvido para auditoria de segurança em servidores baseados no sistema operacional GNU/Linux. A auditoria é feita de acordo com a norma de segurança NBR ISO/IEC 27002 que possibilita às organizações a implementação de um Sistema de Gestão da Segurança da Informação (SGSI), através do estabelecimento de uma política de segurança, controles e gerenciamento de riscos (CUGIK, 2007).



Fonte: Cugik (2007, p. 57).

Figura 4 - Software para verificação de conformidade de servidores GNU/Linux à norma de segurança NBR ISO/IEC 27002

O software foi desenvolvido em PHP e permite que verificações da norma NBR ISO/IEC 27002 sejam adicionadas e removidas, permitindo o auditor, configurar a auditoria para cada tipo de servidor (CUGIK, 2007).

2.5.2 Software para avaliação da segurança da informação de uma empresa conforme a Norma NBR ISO/IEC 17799

Este software tem por objetivo fazer a avaliação da adequação de uma empresa conforme a norma NBR ISO/IEC 17799, conforme apresentado na Figura 5. A norma NBR ISO 17799 aborda a segurança da informação da organização (ROSSEMANN, 2002).

Fonte: Rossemann (2002, p. 35).

Figura 5 - Software para avaliação da segurança da informação de uma empresa conforme a Norma NBR ISO/IEC 17799

As principais características do software são:

a) possui um *check list* para verificar o grau de segurança da informação na empresa avaliada;
b) cada pergunta do *check list* possui um grau de importância dentro de uma organização;
c) no final da avaliação é obtida uma nota, a partir da média ponderada das notas fornecidas a cada uma das perguntas.

2.5.3 Fortify SCA

É uma ferramenta comercial desenvolvida pela FORTIFY que possibilita a identificação de vulnerabilidades em um código fonte. Ajuda os desenvolvedores de software a encontrar problemas de vulnerabilidades no ciclo de desenvolvimento (BRAZ, 2008).

A análise é feita de acordo com as regras estabelecidas pela própria FORTIFY chamada de *rulepack*. Ele suporta a análise nas linguagens Java, .NET, PHP, ColdFusion, JavaScript e PL-SQL. Foi desenvolvido em Java e por isso funciona em várias plataformas (BRAZ, 2008).

3 DESENVOLVIMENTO

Este capítulo contém a especificação, os requisitos, casos de uso e diagramas de classe do software desenvolvido e a operacionalidade do sistema.

3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

Segue os principais requisitos do software, que foram elaborados a partir de um modelo de auditoria e baseado nos requisitos da norma ISO/IEC 15408:

a) o sistema deve fornecer uma lista com todos os requisitos da norma ISO/IEC 15408 para o auditor selecionar quais destes requisitos serão analisados (Requisito Funcional - RF);
b) o sistema deve permitir a entrada do nome do auditor que fará a auditoria (RF);
c) o sistema deve permitir a entrada da instituição a ser avaliada como também o produto a ser avaliado (RF);
d) o sistema deve fazer a auditoria automática de acordo com os requisitos selecionados pelo auditor (RF);
e) o sistema deve permitir adicionar plugins[3] para futuras extensões permitindo a implementação automática dos outros requisitos não automatizados (RF);
f) o sistema deve fornecer uma biblioteca de extensão (RF);
g) o sistema deve permitir exibir no final da auditoria, um *check list* com os requisitos não automatizados para que o auditor possa fazer a auditoria manual desses requisitos e armazenar suas observações (RF);
h) o sistema deve gerar um relatório com as informações encontradas e as fornecidas pelo auditor (RF);
i) o sistema deve realizar a auditoria em sistemas desenvolvidos na linguagem JAVA (RF);
j) o sistema deve analisar automaticamente requisitos de proteção de dados do usuário, auditoria de segurança e identificação e autenticação (RF);

[3] *Plugin* é um arquivo que fornece recursos adicionais a um programa ou aplicativo

k) o sistema deve ser desenvolvido usando orientação a objeto (Requisito Não Funcional – RNF);
l) o sistema deve ser implementado na linguagem de programação Java (RNF).

3.2 ESPECIFICAÇÃO

Para a especificação do software foram utilizados os diagramas da *Unified Model Language* (UML) como caso de uso, diagrama de classe e diagrama de sequência. Para montar a especificação foi utilizada a ferramenta Enterprise Architect.

3.2.1 Diagrama de Casos de Uso

A seguir são apresentados os diagramas de dois pacotes: criação do plano de auditoria e realização da auditoria, com seus respectivos Casos de Uso (UC), que representam as funcionalidades da ferramenta.

Na Figura 6 é exibido o primeiro pacote, que apresenta os UC, de criação de um novo plano de auditoria. Para poder prosseguir na utilização do sistema deve-se ter um plano de auditoria criado.

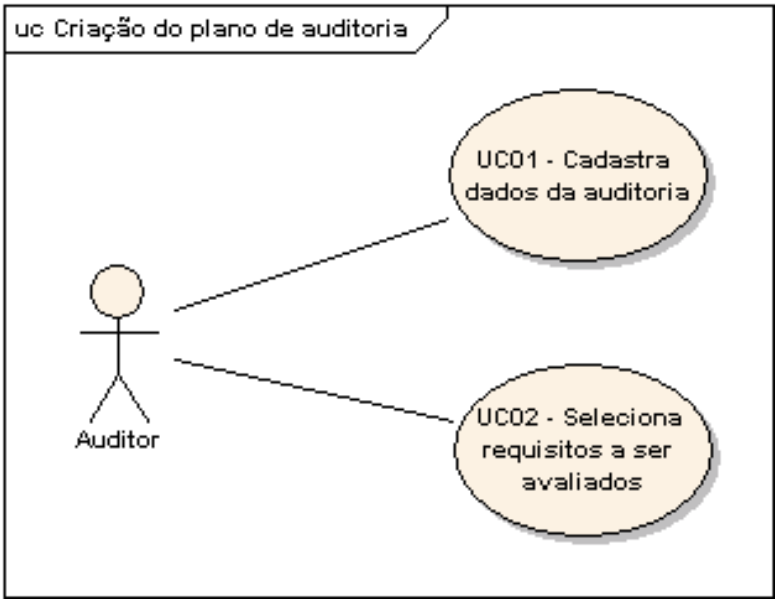


Figura 6 – Caso de uso criação do plano de auditoria

Apresenta-se nos quadros 1e 2 o detalhamento de cada UC pertencente ao pacote criação do plano de auditoria.

UC01 – Cadastra dados da auditoria	
Pré-condições	Ter criado um novo plano de auditoria ou ter aberto um plano existente.
Cenário principal	1) auditor cria um novo plano de auditoria 2) software exibe o cadastro da auditoria 2) auditor cadastra dados da auditoria 4) software mantém informações da auditoria
Cenário alternativo	No passo 1, o auditor pode abrir um plano de auditoria existente 1) software exibe o cadastro de auditoria do sistema 2) auditor modifica informação dos dados da auditoria 3) software altera cadastro da auditoria
Pós-condições	Os dados da auditoria são cadastrados.

Quadro 1 - Detalhamento do caso de uso `Cadastra dados da auditoria`

UC02 – Seleciona requisitos a serem avaliados	
Pré-condições	Ter criado um novo plano de auditoria ou ter aberto um plano existente. Ter a especificação do sistema com os requisitos de segurança que são relevantes para o sistema.
Cenário principal	1) auditor abre a tela de requisitos 2) software exibe a lista de requisitos 2) auditor seleciona os requisitos especificados que serão auditados

Quadro 2 – Detalhamento do caso de uso `Seleciona requisitos a serem avaliados`

Na Figura 7 é exibido o segundo pacote, realização da auditoria, que define os processos para executar a auditoria em um sistema.

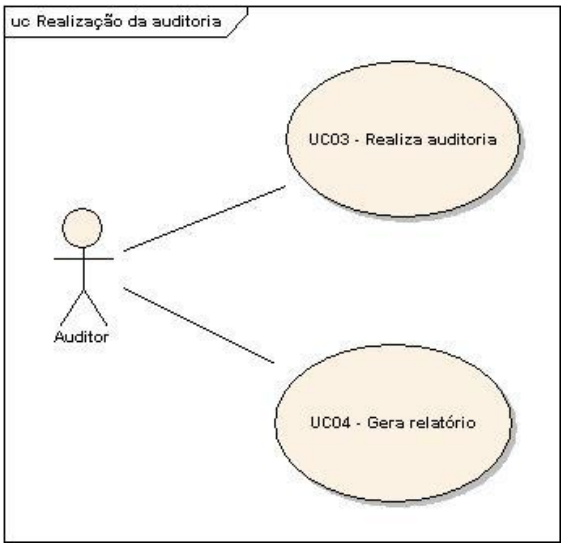


Figura 7 - Caso de uso realização da auditoria

Nos quadros 3 e 4 são detalhados os casos de uso do pacote realização da auditoria:

UC03 – Realiza auditoria automática	
Pré-condições	Ter informado quais requisitos serão verificados e ter ao menos um plugin para a realização do mesmo.
Cenário principal	1) o auditor clica no botão executar auditoria 2) o software carrega os requisitos que possuem forma automática implementada 3) o software executa automaticamente os requisitos automáticos especificados 4) o software finaliza a auditoria 5) o software exibe os requisitos para que sejam auditados 6) O auditor faz a auditoria
Cenário alternativo	No passo 3, durante a execução da auditoria automática, pode ocorrer que necessite alguma interação com o auditor, pedindo alguma informação adicional para a realização da auditoria.
Cenário alternativo	No passo 6 o auditor analisa os requisitos que já foram auditado automaticamente.
Pós-condições	Ter executado o teste.

Quadro 3 – Caso de uso `Realiza auditoria automática`

UC04 – Gera relatório	
Pré-condições	Ter realizado um tipo de auditoria.
Cenário principal	1) auditor avalia auditoria e remove mensagens do campo de observação que não deve ir para o relatório; 2) auditor clica na opção de gerar relatório 3) software exibe tela pedindo o nome do relatório 4) auditor insere o nome do relatório 5) auditor confirma o nome do relatório 4) software gera relatório
Pós-condições	Exibe o relatório com as informações da auditoria.

Quadro 4 – Caso de uso `Gera relatório`

3.2.2 Arquitetura do Software

Nesta seção será apresentada a arquitetura do software desenvolvido, como também a definição dos termos utilizados neste trabalho para melhor entendimento.

A arquitetura é composta por cinco elementos, o software, a API, o Plugin, as Interfaces e o sistema auditado. Na Figura 8 é demonstrada a arquitetura.

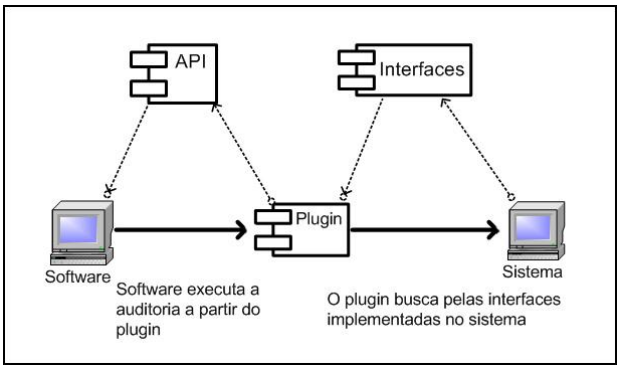


Figura 8 – Arquitetura do Software desenvolvido

Para a execução da auditoria automatizada, a arquitetura do software contem os seguintes elementos envolvidos na arquitetura:

a) `Software`: é o software desenvolvido que contem toda a estrutura para fazer a auditoria. Porém, para se realizar a auditoria automatizada, o auditor deve desenvolver os seus próprios `plugins` com os testes automatizados.
b) `API`: É um conjunto de classes disponibilizado pelo software para o desenvolvimento de `plugins`.
c) `Plugins`: São os testes desenvolvidos pelo auditor para cada requisito utilizado a API. Espera-se que ele tenha desenvolvido os plugins a partir da `API`. Neste `plugin`, o desenvolvedor poderá fazer diversos tipos de testes e acrescentar classe auxiliares para poder exercer a auditoria automatizada.
d) `Interface`: Para poder testar o sistema, espera-se que ele implemente determinadas interfaces. Assim, o `plugin`, desenvolvido pelo auditor, poderá fazer o seu teste baseado nessas interfaces, ou seja, executando esta `interface` que contem uma assinatura que ele possa conhecer no sistema auditado para fazer o teste. O auditor deve fornecer as `interfaces` para o sistema auditado implementar.
e) `Sistema`: É o sistema auditado. Ele deve implementar as `interfaces` disponibilizadas pelo auditor.

3.2.3 Diagrama de classe

Nesta seção são apresentadas as classes utilizadas no desenvolvimento do software e seus relacionamentos. Estas classes são apresentadas de acordo com a arquitetura do software, em quatro pacotes, que são: `software`, `api`, `plugin`, `interface`.

As classes contidas no pacote `software` são parte principal do funcionamento do software desenvolvido. As classes contidas no pacote `api` são responsáveis em dar subsídios para que os auditores possam desenvolver os *plugins* que serão responsáveis pela análise automática dos requisitos do software a ser avaliado. As classes no pacote `plugin`, são as classes utilizadas no plugin construído neste trabalho como exemplo. Já no pacote `interface` são exibidas as `interfaces` que o sistema auditado deve implementar.

Na Figura 9 é mostrado o diagrama de classes correspondente ao pacote `software`.

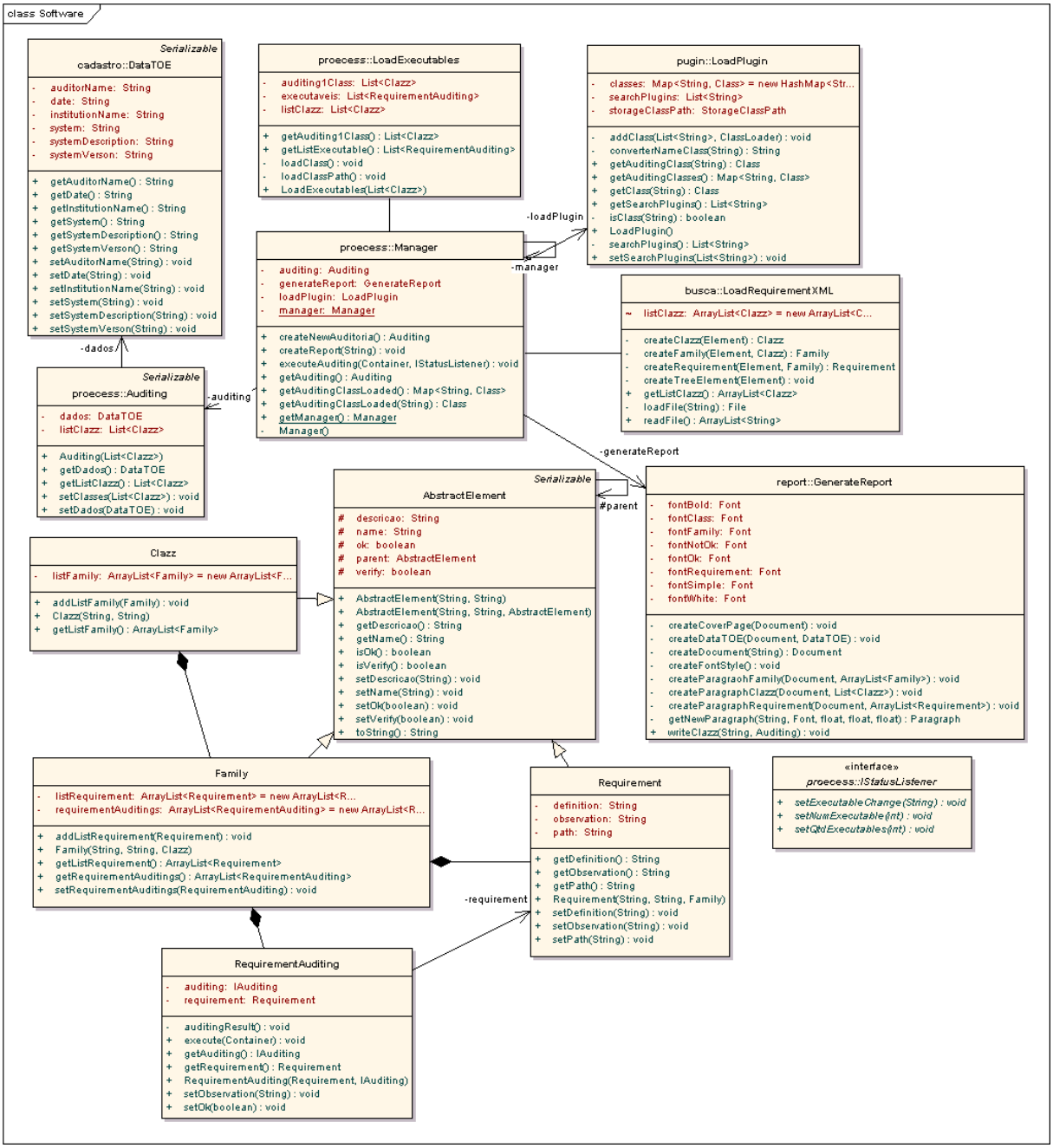


Figura 9 – Diagrama de classe correspondente ao pacote `Software`

Segue o detalhamento das classes contidas no pacote `software`:

a) `Manager`: esta classe é responsável pela criação de um plano de auditoria e por

manter a instância do plano acessível. Também é responsável pelo carregamento dos *plugins* para tratamento específico de cada requisito a ser auditado. Somente irá existir uma instância da classe `Manager` por programa aberto;

b) `Auditing`: esta classe é responsável pela representação de um plano de auditoria e contém todos os requisitos existes da norma e o seu estado atual;
c) `AbstractElement`: esta classe é abstrata e contém os elementos comuns as classes `Clazz`, `Family` e `Requirement`, para qual as informações de nome e descrição são estendidas;
d) `Clazz`: esta classe é responsável por representar uma classe da norma ISO/IEC 15408, contendo todas as famílias pertencentes à classe especificada;
e) `Family`: esta classe é responsável por representar uma família da Norma ISO/IEC 15408, contendo todos os requisitos pertencentes à família especificada;
f) `Requirement`: esta classe é responsável por representar um requisito da norma ISO/IEC 15408. Cada requisito guarda a informação se deve ser auditado ou não, através método `setVerify`;
g) `IStatusListener`: esta classe é responsável por monitorar e informar qual requisito está sendo executado.
h) `DataToe`: esta classe é responsável por armazenar o cadastro da auditoria, como nome do sistema, versão, nome do auditor;
i) `LoadExecutables`: esta classe é responsável por carregar os testes automatizados para que sejam executados;
j) `LoadPlugin`: esta classe é responsável por carregar os *plugins* e colocar eles no *classpath* do software;
k) `GenerateReport`: esta classe é responsável por gerar o relatório;
l) `LoadRequirementXML`: esta classe é responsável por carregar os requisitos da norma e criar instancias das classes, famílias e requisitos;
m) `RequirementAuditing`: esta classe irá conter uma instancia do teste automatizado caso esse requisito seja automatizado.

Na Figura 10 é mostrado o diagrama de classes correspondente ao pacote `API`.

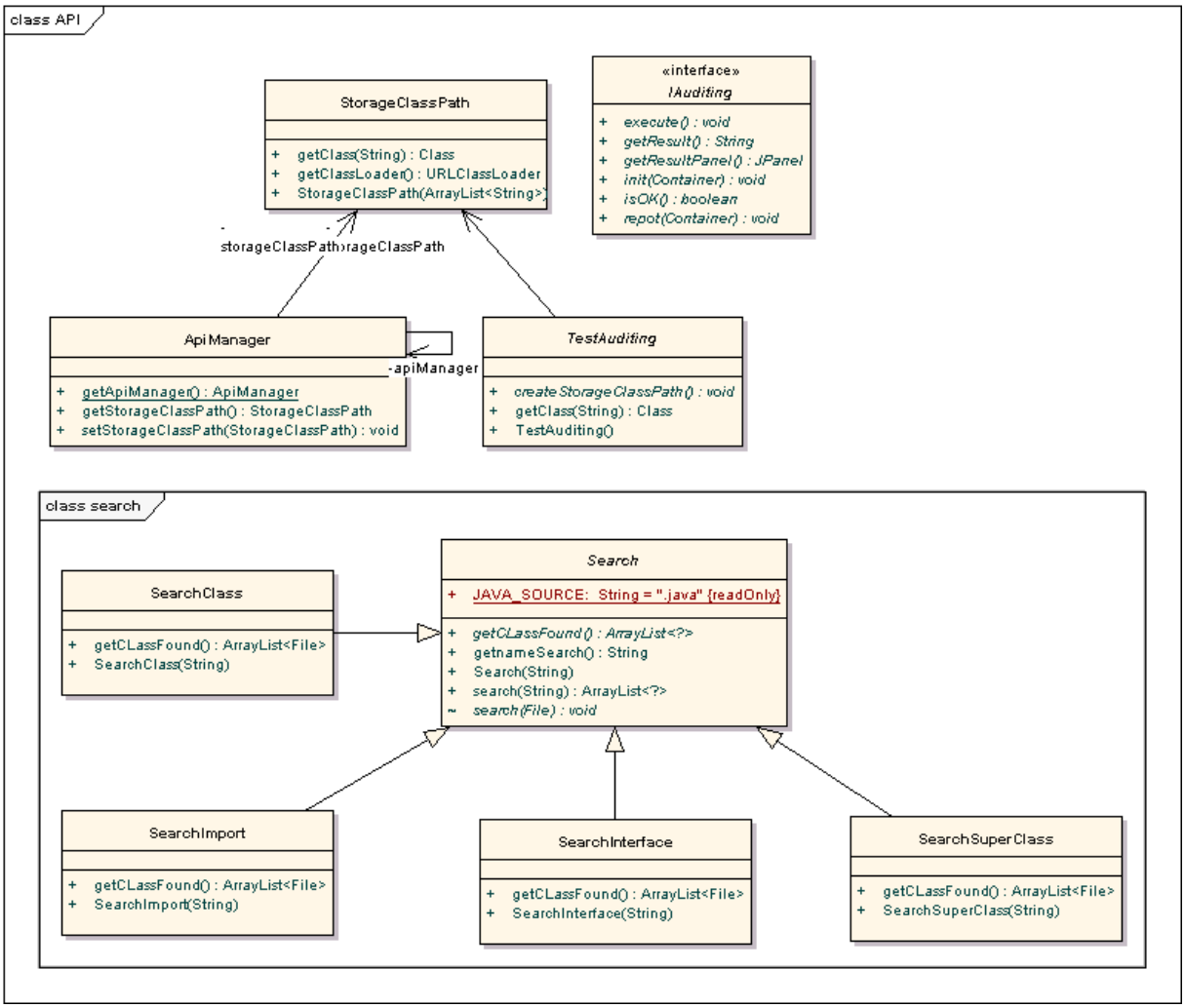


Figura 10 – Diagrama de classe correspondente ao pacote `API`

Segue o detalhamento das classes contidas no pacote `API`:

a) `ApiManager`: esta classe tem como principal objetivo armazenar os `classpath` do sistema auditado. Somente irá existir uma instância da classe `ApiManager` por programa aberto;

b) `IAuditing`: esta interface é responsável por dar subsídios para o desenvolvimento de *plugins* que realizarão testes automatizados no sistema auditado;

c) `TestAuditing`: esta classe é responsável por dar subsídios aos teste implementado quando ele necessitar carregar classes do sistema;

d) `StorageClassPath`: esta classe é responsável por carregar todas as classes do sistema auditado para a realização de testes e também os *plugins* com os testes implementados;

e) Search: está classe é responsável por manter a estrutura para se procurar algum elemento da classe Java em um fonte Java;

f) SearchClass: está classe é responsável por procurar o nome da classe nos fontes Javas;

g) SearchSuperClass: está classe é responsável por procurar nos fontes Javas determinadas classe que elas estendem;

h) SearchInterface: está classe é responsável por procurar nos fontes Javas determinadas interfaces que elas implementam;

i) SeachImport: está classe é responsável por procurar nos fontes Javas determinados *imports*;

Na Figura 11 é mostrado o diagrama de classes correspondente ao pacote `Plugin`.

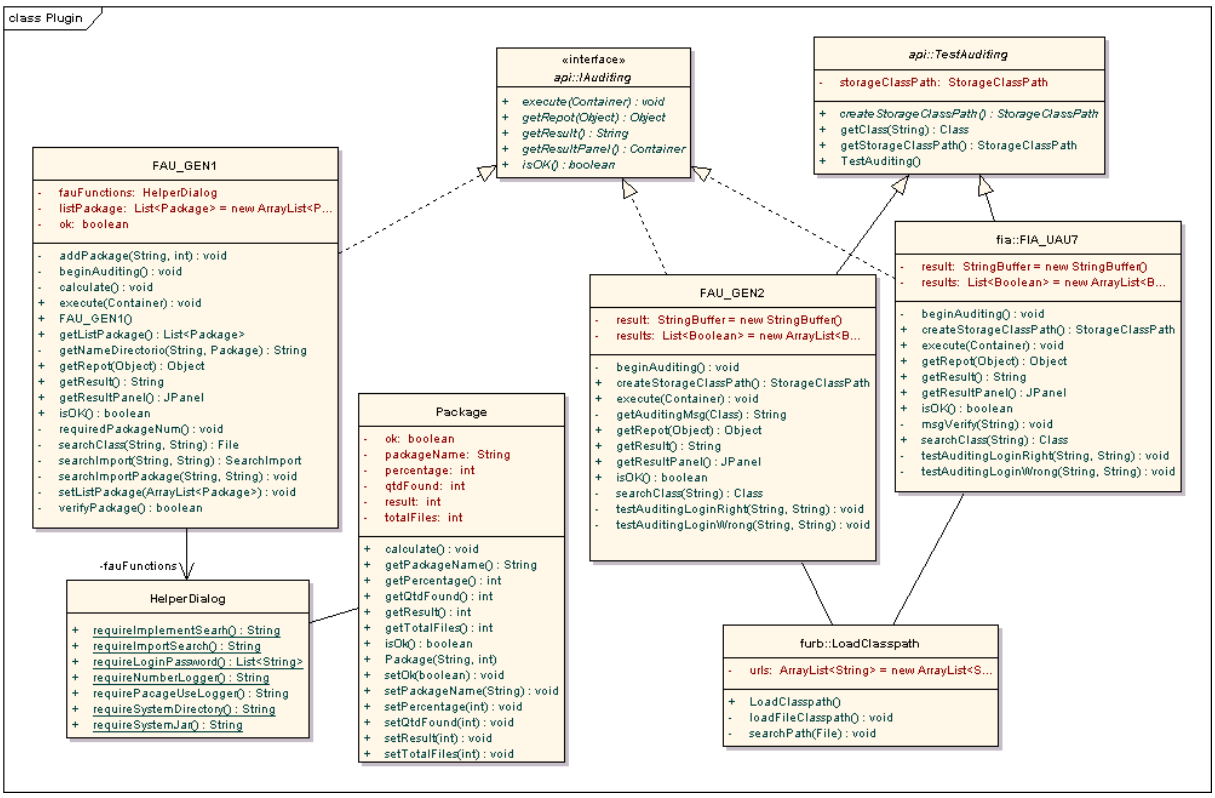


Figura 11 – Diagrama de classe correspondente ao pacote `Plugin`

Segue o detalhamento das classes contidas no pacote `Plugin`:

a) `IAuditing`: Interface da `API` disponibilizada pelo software;

b) `TestAuditing`: classes da `API` disponibilizada pelo software para executar testes automatizados;

c) `FAU_GEN1`: classe que implementa teste automatizado para o requisito Fau_gen.1;

d) `Functions`: classe que contem funções utilizada pela classe `FAU_GEN1`;

e) `Package`: classe utilizada pela `Functions` que guardara informações da auditoria;

f) `FAU_GEN2`: classe que implementa teste automatizado para o requisito Fau_gen.2;

g) `FIA_UAU7`: classe que implementa teste automatizado para o requisito Fia_uau.7;

h) `LoadClasspath`: classe responsável por pedir e carregar o `classpath` do sistema auditado.

Na Figura 12 são mostradas as classes do pacote `interfaces`:

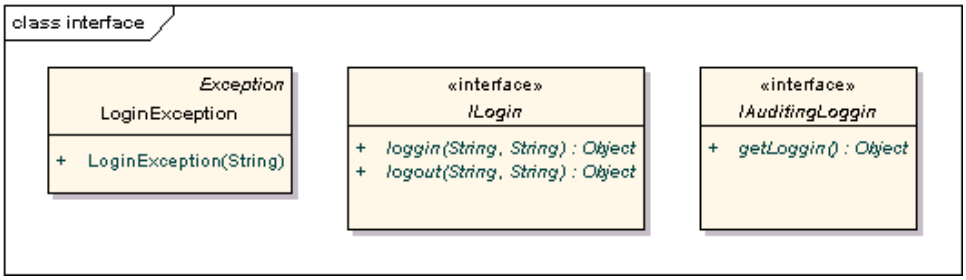


Figura 12 – Diagrama de classe correspondente ao pacote `interface`

Segue a descrição de cada classe contida no pacote interface:

a) `ILogin`: Interface que provê métodos para a execução do *login* do sistema, assim, testes que precisam efetuar *login* e testar esta funcionalidade, farão uso desta interface para executar o teste;

b) `IAuditingLoggin`: Interface que retorna o resultado do *log* da auditoria do sistema, para que seja testado. As classes que necessitam de auditoria devem, implementar esta interface;

c) `LoginException`: utilizada pela interface `ILogin`, para propagar erros na hora da execução do *login*.

3.2.4 Diagrama de seqüência

Esta seção apresenta o diagrama de sequência que representa o conjunto de passos que o programa executa para realizar determinada tarefa, com base nas ações do usuário.

A Figura 13 exibe o diagrama de sequência que representa a abertura do software e como são carregados os *plugins* necessários para a execução da auditoria automatizada.

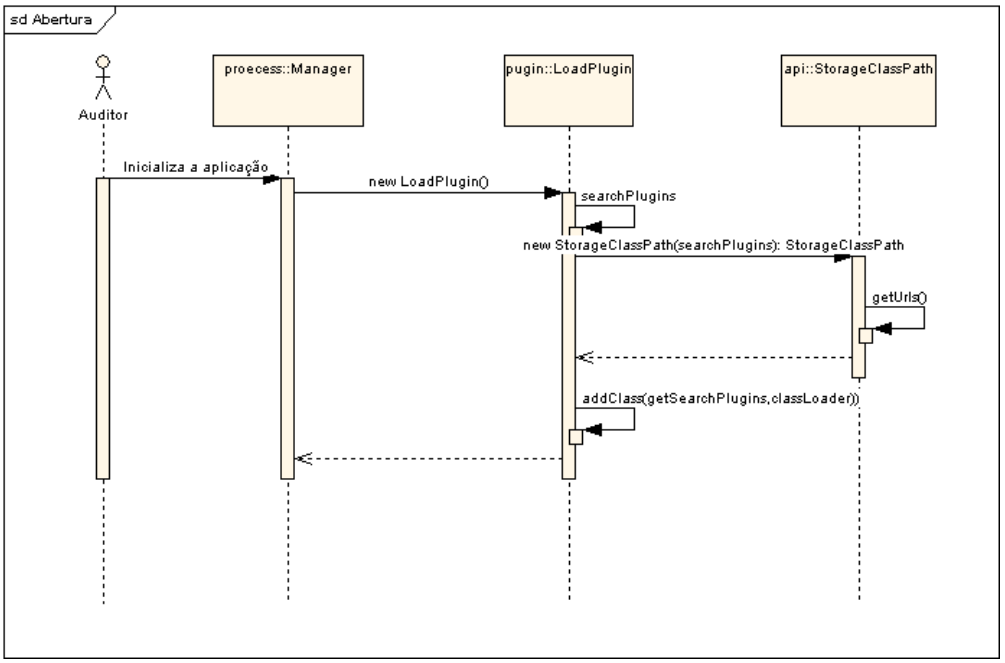


Figura 13 – Diagrama de sequência da abertura do software

A Figura 14 exibe o diagrama de sequência que representa a ação do auditor para executar a auditoria automática. Este diagrama representa as ações do caso de uso realizar auditoria automática.

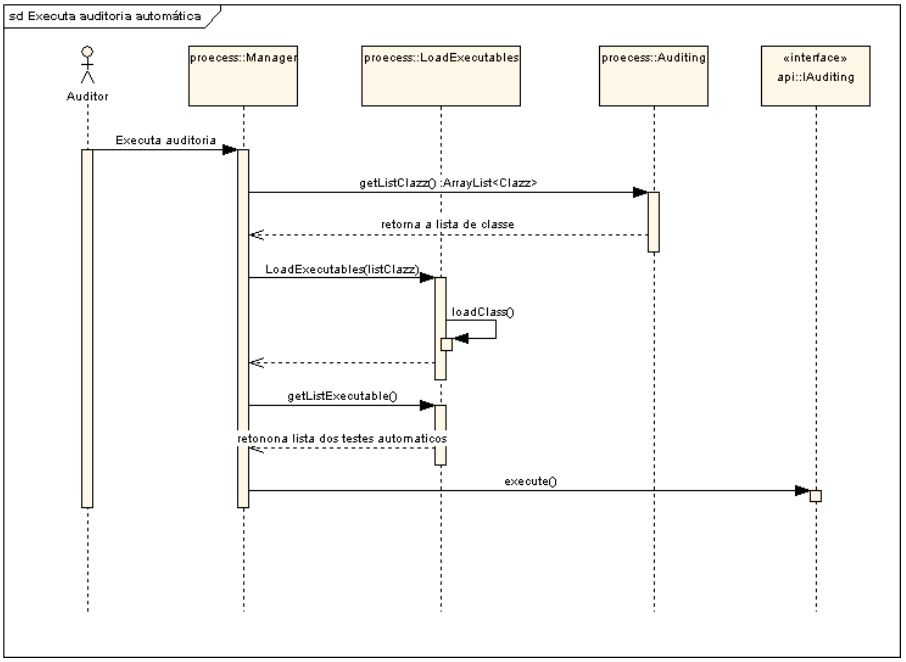


Figura 14 – Diagrama de sequência para executar a auditoria automática

3.3 IMPLEMENTAÇÃO

Nesta seção serão mostradas as técnicas e ferramentas utilizadas e a operacionalidade do software.

3.3.1 Técnicas e ferramentas utilizadas

O protótipo foi desenvolvido na linguagem Java, seguindo o paradigma da orientação a objetos. Foi desenvolvido utilizado o ambiente de desenvolvimento Eclipse – Galileu.

3.3.2 Dom4j

O dom4j é uma biblioteca *open source* para trabalhar com XML, XPath e XSLT na plataforma Java usando o *Java Collections Framework* e suporte para DOM, SAX e JAXP.

Todos os requisitos da norma ISO/IEC 15408 são persistidos em um arquivo XML que é lido pelo software a cada novo plano de auditoria.

No Quadro 5 é apresentado parte da estrutura do arquivo XML.

```
<classe id="FAU" descricao="AUDITORIA DE SEGURANÇA">
   <familia id="FAU_ARP"
            descricao="FAU_ARP ? Resposta automática a auditoria">
      <requisito id="FAU_ARP.">
         FAU_ARP.1 - Alarmes de segurança
      </requisito>
   </familia>
</classe>
```

Quadro 5 – Estrutura do arquivo XML de armazenamento da árvore dos requisitos

Para manter a organização do arquivo XML foi utilizada a classificação disponibilizada pela norma, onde, há classes da norma, que estão divididas em famílias da norma e que possuem os seus requisitos específicos da norma.

Cada atributo utilizado no arquivo XML é representado por uma classe Java. Estas classes Java seguem a mesma hierarquia estabelecida no arquivo XML, ou seja, uma classe da norma contém como filhos um conjunto de famílias da norma, que por sua vez, possuem como filhos, um conjunto de requisitos específicos da norma.

A classe responsável por ler os atributos XML e montar a estrutura hierárquica no Java

é a classe `LoadXML`. No Quadro 6 é exibido parte do código que lê o arquivo XML.

```
public class LoadRequirementXML {

    ArrayList<Clazz> listClazz = new ArrayList<Clazz>();

    /*Itera por todos os nós do XML e cria um novo objeto
    de acordo com o atributo lido*/
    private void createTreeElement(Element element) {
        for (Iterator<Element> iterator = element.elementIterator();
            iterator.hasNext();) {

            Element classElement = iterator.next();

            if (classElement.getName().equals("classe")) {
                Clazz clazz = createClazz(classElement);
                listClazz.add(clazz);

                for (Iterator<Element>
                    iteFam = classElement.elementIterator();
                        iteFam.hasNext();) {

                    Element famElement = iteFam.next();
                    Family family = createFamily(famElement, clazz);
                    clazz.addFamilias(family);

                    for (Iterator<Element>
                        iteReq = famElement.elementIterator();
                            iteReq.hasNext();) {

                        Element reqElement = iteReq.next();
                        Requirement requirement =
                            createRequirement(reqElement, family);
                        family.addRequisito(requirement);
                    }
                }
            }
        }
    }
}
```

Quadro 6 - Parte do código da classe `LoadXML`

3.3.3 QDox

QDox é um analisador de código que extrai definições de classes, interfaces e métodos em códigos fontes que contém a tag @ do JavaDoc.

Neste trabalho a biblioteca QDox foi utilizada com o objetivo de analisar os códigos fontes Java do sistema auditado e assim retornar a estrutura da classe encontrada, podendo verificar se ela implementa, estende ou importa uma determinada constante.

No código apresentado no Quadro 7, o método `search` recebe como parâmetro um arquivo e verifica se ele é um fonte Java. Caso seja um fonte Java, ele cria uma instância de `JavaDocBuilder`, que analisa o código fonte para disponibilizar a informação na forma de uma classe Java.

```
protected void search(File file) {
        if (!file.getName().contains(JAVA_SOURCE)) {
            return;
        }
        try {
            JavaDocBuilder n = new JavaDocBuilder();
            n.addSource(file);
            JavaSource javaSource = n.getSources()[0];
            for (JavaClass jClass : javaSource.getClasses()) {
                inter: for (JavaClass interClass :
                    jClass.getImplementedInterfaces()) {
                    for (String find : interfaceNames) {
                        if (interClass.getName().equals(find)) {
                            getCLassFound().add(file);
                            break inter;
                        }
                    }
                }
            }
        } catch (FileNotFoundException e) {
            throw new RuntimeException(e);
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }
```

Quadro 7 – Exemplo de código que procura um fonte Java com uma determinada interface

3.3.4 iText

O iText é uma biblioteca que permite manipular dinamicamente arquivos PDF no Java. Desta forma, o desenvolvedor constrói seus aplicativos automatizando o processo de criação e manipulação dos arquivos PDF.

Como ela é de fácil manipulação, foi escolhida para a geração do relatório final, visto que, o relatório não é tão complexo e não necessita de APIs especificas para sua geração. No Quadro 8 é exibido parte do código que realiza a geração do relatório final no formato PDF.

```
    public Document createDocument(String path)
       throws FileNotFoundException, DocumentException {
        Document document = new Document(PageSize.A4, 50, 50, 50, 50);
        //cria o documento
        PdfWriter.getInstance(document, new FileOutputStream(path));
        document.open();
        return document;
    }

    public void writeClazz(String path, Auditing auditing)
       throws FileNotFoundException, DocumentException {
                    Document document;
            createFontStyle();
            document = createDocument(path);
            createDataTOE(document, auditing.getDados());
            createParagraphClazz(document, auditing.getListClazz());
    }
```

Quadro 8 - Parte do código de geração do relatório final

3.3.5 Desenvolvimento do plano de auditoria

Quando se inicializa o software, ele instancia um único objeto `Manager` que será o responsável por delegar o carregamento dos *plugins* adicionados a ele e gerenciar cada novo plano de auditoria.

Para o carregamento dos *plugins* o `Manager` irá chamar a classe LoadPlugin que irá carregar os *plugins*, do diretório da aplicação `\plugins`, para o `classpath` do software, para então, ser executado.

A classe `LoadPlugin` vai instanciar um novo objeto da classe `StorageClassPath` e passará para ela o caminho dos *plugins* a serem carregados pelo `classpath` criado por ela. Também é responsável por procurar as classes que implementam a interface `IAuditing`. Quando encontrar uma classe que implementa a classe `IAuditing` a mesma será mantida em uma lista para que, no momento de sua execução, não seja necessário realizar uma nova procura. No Quadro 9 é exibido parte do código que carrega as classes que implementam a interface `IAuditing`.

```
public class LoadPlugin {

    private void addClass(ArrayList<String> path, ClassLoader loader){

        for (String url : path) {
            JarFile jarFile;
            jarFile = new JarFile(url);
            Enumeration<JarEntry> entries = jarFile.entries();
            while (entries.hasMoreElements()) {
                JarEntry jarEntry = (JarEntry)
                    entries.nextElement();
                String name = jarEntry.getName();
                if (name.length() > 6 && isClass(name)) {
                    Class clazz =
                        Class.forName(converterNameClass(name), true,
                                loader);
                    if (clazz.newInstance() instanceof IAuditing) {
                        getAuditingClasses().put(clazz.getSimpleName(),
                                                clazz);
                    }
                }

            }
        }
    }
}
```

Quadro 9 – Parte do código que carrega as classes que implementam a interface `IAuditing`

3.3.6 Desenvolvimento da API para *plugin*

O software somente irá executar uma auditoria automática quando forem inseridos *plugins* com a implementação dos testes automatizados.

Para desenvolver os *plugins* o software fornece uma API para o desenvolvedor implementar testes automatizados para cada requisito da norma. O desenvolvedor poderá escolher as interfaces que ele quer auditar, por exemplo, se ele for auditar um sistema que utiliza a estrutura de segurança JAAS, então, deve-se utilizar testes esperando os comportamentos necessários para essa implementação.

Os testes podem fornecer interação com auditor, como pedir alguma informação, como o `classpath` da aplicação.

A API possui duas classes principais, `IAuditing` e `TestAuditing`. `IAuditing` é uma interface que todos os *plugins* devem implementar, para que o software possa interagir e acoplar a classe implementada com o sistema. No Quadro 10 é exibido o código da interface `IAuditing`.

```
public interface IAuditing {

    void execute(Container container);

    boolean isOK();

    Container getResultPanel();

    String getResult();

    Object getReport();
}
```

Quadro 10 – Interface `IAuditing`

O método `execute` é chamado pelo software para iniciar a auditoria do requisito. Neste método será feito o teste do requisito, acabado de executar este teste, passa-se para o próximo requisito.

O método `isOk` retorna para o software se as verificações foram feitas com sucesso ou não.

O método `getResultPanel` é um método opcional, dependendo de como ocorreu a auditoria e quais elementos foram envolvidos, o teste poderá fornecer um resultado mais detalhado para o auditor, como erros que ocorreram durante a execução, se foram vários métodos executados quais tiveram sucesso, quais falharam.

O método `getResult` somente retorna uma string como resposta para ser colocada na observação.

Outra classe que faz parte da API é a `TestAuditing`, quando o teste precisar carregar as classes do sistema para executar parte da rotina do sistema, o teste deve estender a classe `TestAuditing`, que fornece o `classloader` para carregar o classloader do sistema.

A única restrição é que o nome da classe de teste deve ser o nome do requisito a ser testado, por exemplo, se o desenvolvedor irá implementar um testes para o requisito FAU-GEN.1 da norma, o sistema espera que ele utilize o nome padrão para esse requisito no nome da sua classe, a referência da sua família e o número do requisito dentro daquela família. Assim o nome da classe exemplo deve-se chamar `FAU_GEN1`.

3.3.7 Exemplo de teste automatizado

Foram implementados testes para a verificação dos resultados do *plugin*. Para a realização dos testes foi utilizado o sistema Scrump desenvolvido por alunos do curso de ciência da computação da FURB na disciplina de projetos de software. O sistema é voltado para auxiliar em um projeto que utiliza o gerenciamento ágil *scrum*, e foi desenvolvido com a tecnologia Google Web Toolkit (GWT).

Para testar o software desenvolvido, foram implementados os requisitos de segurança no sistema FAU_GEN.1- Geração de dados para auditoria, FAU_GEN.2 - Associação do usuário ao evento de auditoria, FIA_UAU.7 - Tratamento de falha de autenticação.

3.3.7.1 Teste FAU_GEN.1

Conforme Albuquerque e Ribeiro (2002, p. 114) o requisito FAU_GEN.1, é responsável por garantir que o sistema gere dados de auditoria, para os principais eventos do sistema como exemplo as conexões de banco de dados.

Para que este requisito fosse testado foi implementado o log de auditoria em algumas classes do sistema scrump, utilizando o próprio recurso do Java `java.util.logging.Logger` Este teste é destinado a testar o fonte do sistema e verificar quais fontes Java importam a classe `java.util.logging.Logger`, deduzindo assim que esta classe contem auditoria.

Na execução do teste é aberta um janela para que o auditor informe dados para que a auditoria seja feita. Os dados que ele deve informar são o caminho dos fontes do sistema, o

caminho do `Logger` utilizado pelo sistema e quais pacotes são os mais críticos que devem conter a auditoria. Para medir o nível de cada pacote o auditor deve informar a porcentagem esperada de classes com auditoria

Para facilitar o armazenamento das informações dos pacotes a serem procuradas, foi criada uma classe Package. Quando for procurado os `imports` no sistema, será armazenado na classe Package quais foram as quantidades encontradas e as quantidades esperadas para o pacote informado. No final, para o resultado da auditoria o método `calculate` da classe auxiliar de FAU, itera por todos os pacotes mandando calcular o resultado como mostra o Quadro 11, verificando a conformidade com a norma.

```
public void calculate() {
		for (Package package1 : getListPackage()) {
			package1.calculate();
		}
	}
```

Quadro 11 – Rotina que itera pela lista de `packege` e executa a conformidade com a norma

Para procurar pelos fontes que importam o `Logger`, foi utilizado a classe `SearchImport` que contem as rotinas para procurar os fontes com determinados `imports` exibido no Quadro 12.

```
public SearchImport searchImport(String importLogger, String directory) {
		SearchImport searchClass = new SearchImport(importLogger);

		searchClass.search(directory);
		if (searchClass.getCLassFound().size() == 0) {
			JOptionPane.showMessageDialog(null, "Não encontrou " +
					"nenhuma classe com o import " +
					importLogger);
		}
		return searchClass;
	}
```

Quadro 12 – Rotina que procura pelo `import` de `Logger`

3.3.7.2 Teste FAU_GEN.2

Conforme Albuquerque e Ribeiro (2002, p. 116) o requisito FAU_GEN.2, além de exigir a auditoria, exige que a auditoria faça a associação da identidade do usuário.

O teste deste requisito é diferente do FAU_GEN.1. Este requisito exige que o usuário que executou a ação esteja inserido no *log* da auditoria. Para que este teste seja realizado foi implementado um teste que carrega e executa uma determinada rotina do sistema que contem a auditoria. Para saber qual classe do sistema testar, foi disponibilizado uma interface que o sistema deve implementar para que se possa fazer o teste. As interfaces disponibilizadas são

`ILogin`, com os métodos `login` e `logoff` e `IAuditingLoggin`, que contem o método `getAuditing`. Assim o teste procura pela classe que implementa estas interfaces.

Primeiramente o teste pede informações para o auditor, como um arquivo que contenha o `classpath` do sistema auditado, um usuário com uma senha válida e um usuário e senha inválidos para que se possa executar o teste. Assim o teste tenta entrar no sistema com uma senha inválida. O sistema deve fazer a auditoria desta ação, ou seja, deve registrar que houve tentativa de entrar no sistema mas não foi possível. Feito isso o teste tentar entrar com o usuário e senha válidos. O sistema deve retornar que o usuário fornecido pelo auditor efetuou a entrada no sistema.

Como este teste executa uma ação no sistema ele deve estender a classe TestAuditing, que exige a implementação do método `createStorageClassPath` para pedir o *classloader* do sistema. Para carregar os `classpath` do sistema foi criado a classe `LoadClasspath` para auxiliar nos testes que necessitam deste recurso. O código exibido no Quadro 13.

```
public class LoadClasspath {

	private ArrayList<String> urls = new ArrayList<String>();

	public LoadClasspath() {
		loadFileClasspath();
		// Cria um novo storegeClassPath passando as urls do sistema para que
		// o classloader ache as classes futuramente
		StorageClassPath storageClassPath = new StorageClassPath(urls);
		ApiManager.getApiManager().setStorageClassPath(storageClassPath);
	}

	// carrega o classpath do sistema, espera-se que ele esteja em um arquivo
	private void loadFileClasspath() {
		String pathname = JOptionPane
				.showInputDialog("Entre com o arquivo do classpath");
		File file = new File(pathname);
		try {
			FileReader reader = new FileReader(file);
			BufferedReader br = new BufferedReader(reader);
			String readLine = br.readLine();
			while (readLine != null) {
				searchPath(new File(readLine));
				readLine = br.readLine();
			}
		} catch (FileNotFoundException e) {
			JOptionPane.showMessageDialog(null,
			"Erro ao ler arquivo com o classpath","", JOptionPane.ERROR_MESSAGE);
		} catch (IOException e) {
			JOptionPane.showMessageDialog(null,
			"Erro ao ler arquivo com o classpath","", JOptionPane.ERROR_MESSAGE);
		}

	}

	private void searchPath(File file) {
		if (file.isDirectory()) {
			for (File lib : file.listFiles()) {
				urls.add(lib.getPath());
			}
		} else {
			urls.add(file.getPath());
		}
	}
}
```

Quadro 13 – Carregamento do classpath do sistema pelo *plugin*

3.3.7.3 Teste FIA_UAU.7

Conforme Albuquerque e Ribeiro (2002, p. 139-140) o requisito FIA_UAU.7 diz respeito a restrição da mensagem caso ocorra erro no login, por exemplo, deve mostrar uma mensagem que o usuário ou a senha estão inválidos, e não somente um.

A classe que testa este requisito é a FIA_UAU.7. Ela também estende de `TestAuditing` e implementa o `IAuditing` para que seja buscada quando for executado a auditoria automática. A maneira de testar este requisito é bem parecido com o FAU_GEN.2, ele busca pelas mesmas interfaces e entra no sistema com o usuário válido e usuário inválido, porém o objetivo deste teste é a verificação da mensagem que retorna para o usuário caso o login ou a senha não estiverem corretos.

Assim tem-se o método que entra no sistema com o usuário ou senha errado. No final da execução do método é aberta uma tela para que o auditor verifique se a mensagem está correta ou não. O código é exibido no Quadro 14.

```
private void testAuditingLoginWrong(String login,
                        String pass) throws ClassNotFoundException,
                        SecurityException, NoSuchMethodException {
                result
                                .append("Mensagem de erro deve ser genérica: \n");
                String msg = "";
                Class<?> forName = searchClass(
                                "furb.br.seguranca.ILogin");
                if (forName == null) {
                        JOptionPane
                                        .showMessageDialog(
                                                        null,
                                                        "Não foi encontrada nenhuma classe " +
                                                        "para a realização da auditoria");
                }
                Method method = forName.getMethod("loggin",
                                new Class[] { String.class, String.class });
                try {
                        Object invoke = method.invoke(forName
                                        .newInstance(), login, pass);
                        results.add(false);
                        result.append("Não deveria ter logado\n");
                } catch (IllegalArgumentException e) {
                        msg = e.getCause().getMessage();
                        e.printStackTrace();
                } catch (IllegalAccessException e) {
                        msg = e.getCause().getMessage();
                        e.printStackTrace();
                } catch (InvocationTargetException e) {
                        // Adiciona a msg do erro do login
                        msg = e.getCause().getMessage();
                        e.printStackTrace();
                } catch (InstantiationException e) {
                        msg = e.getCause().getMessage();
                        e.printStackTrace();
                }

                result.append("\n" + msg);
                // ok
                msgVerify(msg);
}
```

Quadro 14 – Rotina de verificação da mensagem do *login*

3.3.8 Operacionalidade da implementação

Nesta seção será apresentada a operacionalidade do software desenvolvido. Quando se abre o software ele deve carregar os *plugins* que o auditor usará para a auditoria. Para começar a auditoria o auditor deve abrir um novo plano de auditoria ou um plano já existente, como exibido na Figura 15.

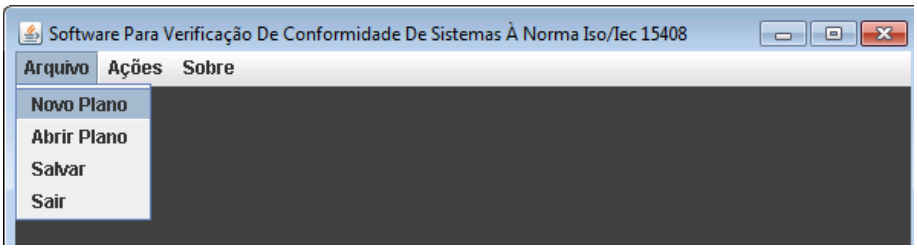


Figura 15 - Criar um novo plano de auditoria

A primeira tela que aparece para o auditor é a tela de especificação, onde o auditor entrará com as informações da auditoria, como nome da instituição, o sistema auditado, a versão do sistema, o nome do auditor, data, e uma breve explicação sobre o sistema, quais seus objetivos, e o que ele presa mais em segurança, conforme apresentado na Figura 16.

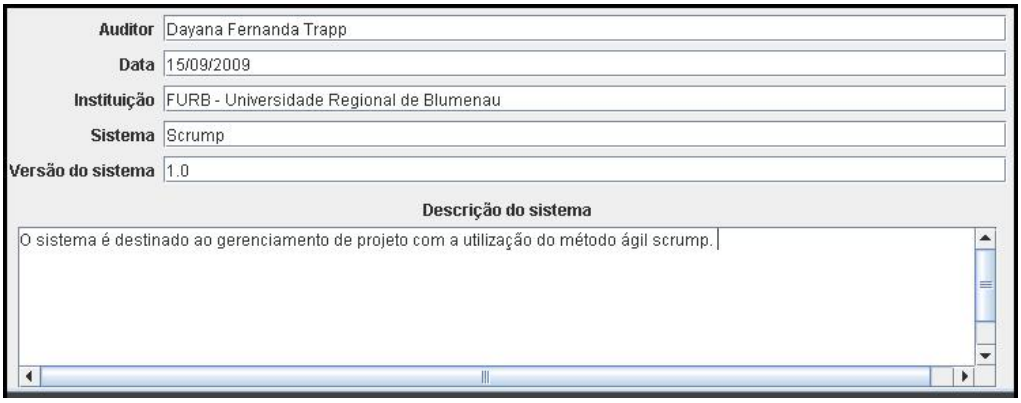


Figura 16 - Cadastro básico do sistema auditado

Após ter cadastrado as informações, o auditor vai para a tela seguinte para selecionar os requisitos que serão avaliados naquele sistema de acordo com a ST definido pelo auditor e a empresa, conforme apresentado na Figura 17.

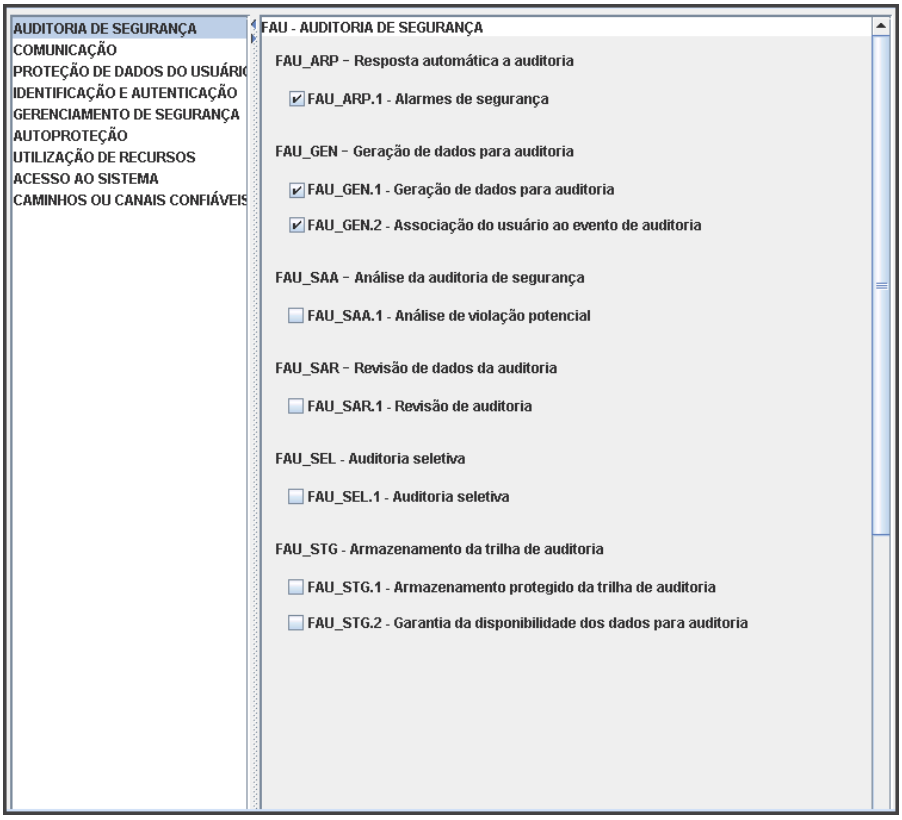


Figura 17 - Seleção dos requisitos a serem auditados

Terminado de selecionar os requisitos que serão avaliados, o auditor começará a fazer a auditoria. Existem dois tipos de auditoria: automática, que possui recursos para auxiliar o auditor durante a auditoria e manual que é padrão para todos os requisitos.

Primeiramente o auditor deverá executar a auditoria automatizada, em ações, executar como mostra a Figura 18. Nesse instante o software procura, nos *plugins*, se existe teste automatizado para os requisitos selecionados e o executa. O auditor sempre deve estar presente quando for executar a auditoria automatizada, pois ela pode pedir informações adicionais, como por exemplo, o `classpath` do sistema auditado.

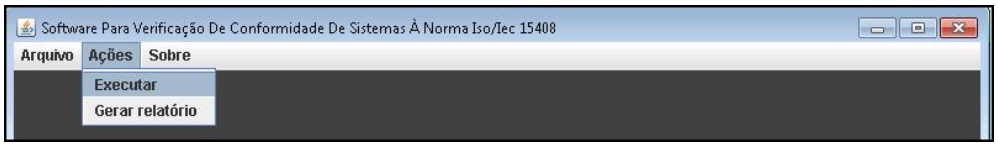


Figura 18 – Execução da auditoria

Enquanto são executados os testes automáticos o auditor não poderá mexer no software, ficando com a tela de status aberta (Figura 19), que dá um retorno para o auditor de quantos testes automatizados foram encontrados e qual está sendo executado.

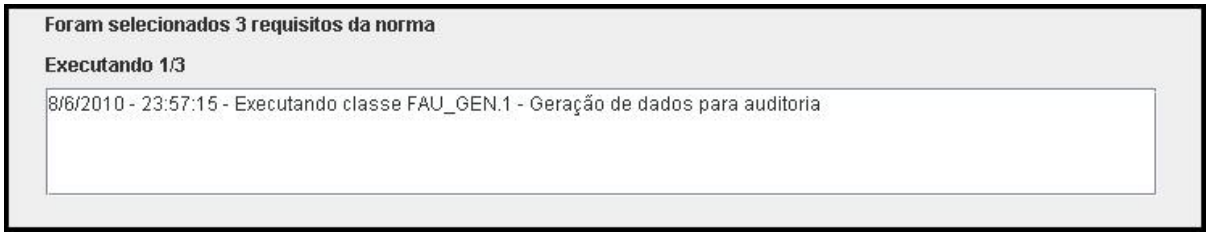


Figura 19 - Tela de execução dos requisitos da norma

Quando os testes terminam de executar, o software apresenta uma tela com todos os requisitos que devem ser avaliados para que o auditor possa fazer a auditoria manual, conforme apresentado na Figura 20. Para manter uma ordem de classe, família e requisito, optou-se em inserir os resultados da auditoria automática junto com a manual, ou seja, o auditor passará de requisito em requisito e verificará que determinado requisito foi executado automaticamente. O requisito já vem com uma resposta da auditoria, como também vai disponibilizar um botão que poderá retornar uma nova janela informações adicionais sobre a análise.

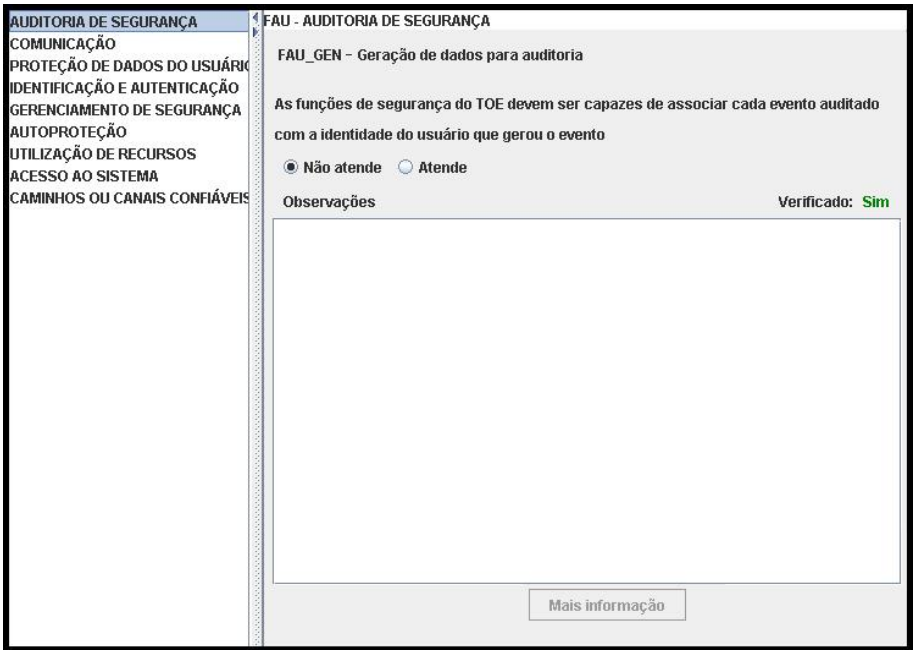


Figura 20 - Resultados da auditoria

Caso tenha esse resultado, ele também será gerado no relatório como informações a mais. O auditor pode analisar o requisito testado pelo software e agregar mais alguma observação no quadro de observações.

Seguindo o plano de execução da auditoria, para os requisitos que foram classificados como não verificados, o auditor somente tem a opção de dizer se o requisito está sendo atendido ou não e colocar uma informação ou observação deste requisito em relação ao sistema.

Terminada de fazer a auditoria, o auditor pode gerar o relatório indo no menu ferramentas, gerar relatório (Figura 21). Ele pode salvar o plano de auditoria para recuperação futura, através da opção arquivo, salvar.

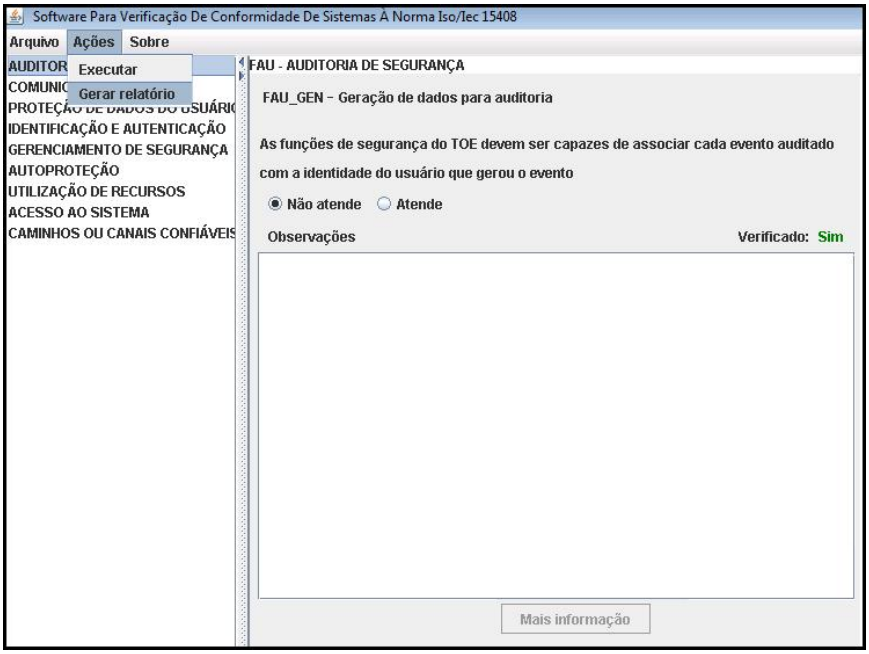


Figura 21 - Gerar relatório da auditoria

O relatório apresenta primeiramente os dados da auditoria (Figura 22).

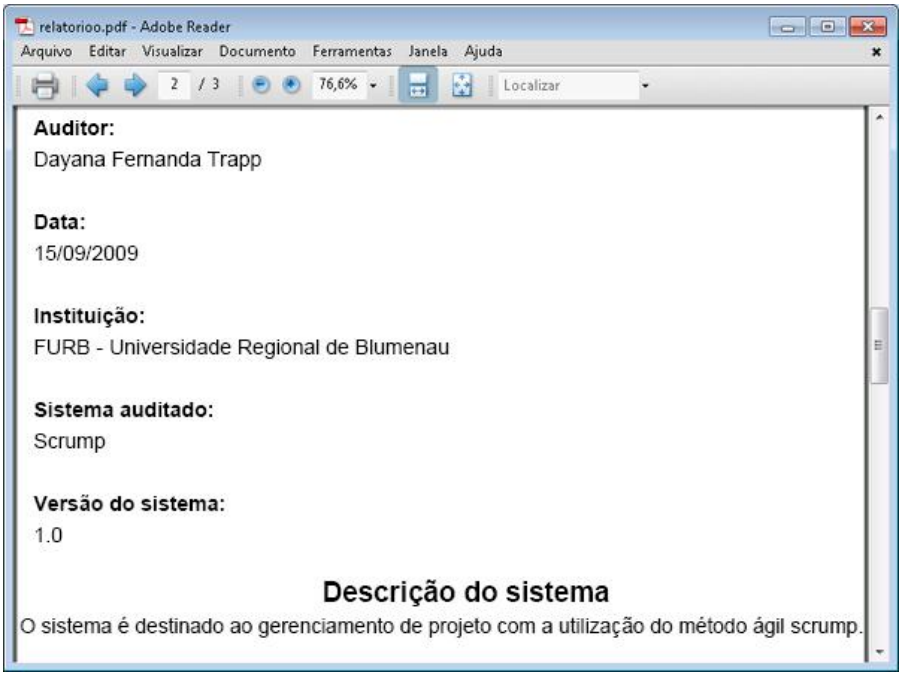


Figura 22 – Relatório com os dados da auditoria

Na Figura 23 é exibido o resultado da auditoria de cada requisito.

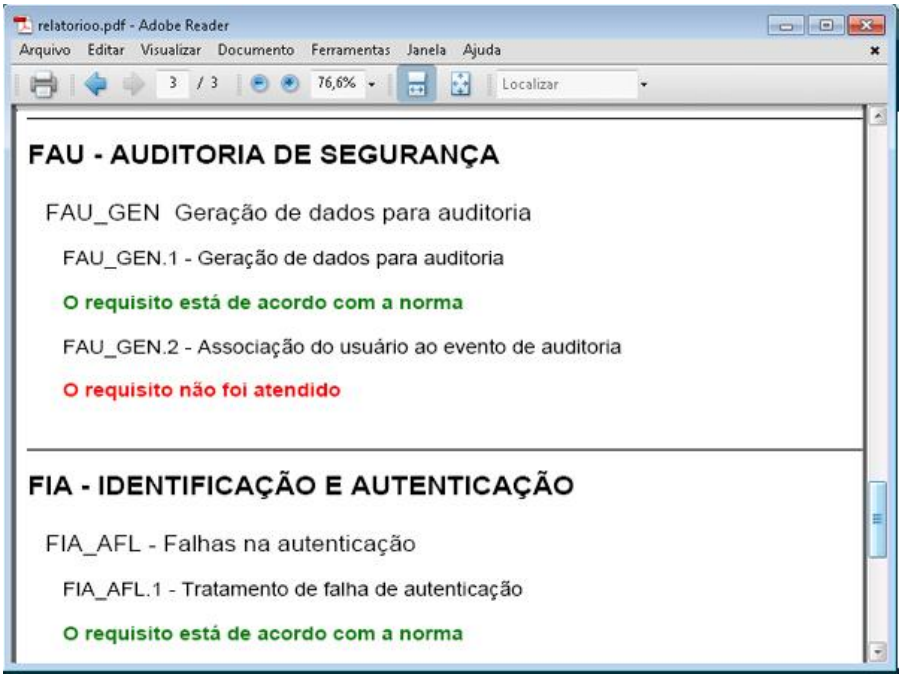


Figura 23 – Relatório com os resultados da auditoria

O relatório apresenta os resultados divididos em grupos de classe, família e então o resultado de cada requisito. Como mostra na figura a classe FAU teve uma família auditada e os dois requisitos dessa família foram auditados. O requisito FAU_GEN.1 foi atendido pela norma, já o FAU_GEN.2 não.

3.4 RESULTADOS E DISCUSSÃO

Com o término deste trabalho pode se verificar que o auditor é capaz de fazer uma auditoria em um sistema conforme a especificação da norma ISO/IEC 15408. Porém, houve uma dificuldade em estabelecer uma regra para a verificação da auditoria, visto que, existem diferentes formas e API para a implementação de segurança. Para contornar o problema foi desenvolvido uma API que o software fornecerá para que o auditor possa desenvolver *plugins* com testes automatizados. Assim, o auditor pode ter um conjunto de *plugins* com diferentes regras de implementação a ser testadas, ou seja, quando for auditar um sistema, o sistema deve implementar determinadas interfaces, o auditor utilizará os *plugins* referentes àquelas interfaces para executar a auditoria.

Outras vantagens foram encontradas ao utilizar *plugins*, como implementar verificações próprias para cada sistema, implementar verificações de testes para outra linguagem de programação, utilizar outros modos de verificação, como a análise estática.

Em relação aos trabalhos correlatos o software desenvolvido pela Fortify é um software utilizado pela própria Fortify e testa vulnerabilidades em um sistema pela análise estática e pelas regras desenvolvida por ela. Em comparação ao software desenvolvido ela não vincula as vulnerabilidades encontradas com os requisitos da norma ISO/IEC 15408. No desenvolvimento deste trabalho não foi utilizado o tipo de teste de análise estática, porém, ele deve ser possível de se fazer, visto que o software oferece uma API para que seja implementado os testes e nada impede que o auditor implemente um teste utilizando a análise estática com suas regras.

Em relação ao "software para avaliação da segurança da informação de uma empresa conforme a norma NBR ISO/IEC 17799" ele não é um software desenvolvido para a norma em questão e não oferece teste automatizado, somente em forma de *check list*. Este trabalho não desenvolveu regras para a implementação de cada requisito da norma, visto que, não era o escopo do trabalho a criação de regras e sim fazer um software para que fosse possível fazer a auditoria.

Já em relação ao "software livre para verificação de adequação de servidores gnu/linux à norma de segurança NBR ISO/IEC 27002" também não faz auditoria da norma em questão, porém este executa auditoria automática, pelos *scripts* desenvolvidos. O software foi desenvolvido para web tornando-o mais prático.

4 CONCLUSÕES

A cada dia novas tecnologias vão surgindo e os computadores estão mais conectados, não somente os computadores, como celulares, *palms*, TV digital, etc. Um computador sem acesso a internet já não faz sentido. Com as novas tecnologias e soluções disponibilizadas crescem também o número de ameaças nos sistemas e a exposição de ataques de *hackers* buscando informações pessoais, como senhas de banco.

A segurança hoje é fundamental no desenvolvimento dos sistemas. As empresas precisam garantir que os seus sistemas são seguros. Mostrar isso para os clientes não é uma tarefa fácil, pois não tem como se provar que um sistema é seguro, porque não existe sistema seguro. Ele é seguro até que se encontre uma vulnerabilidade. Mas, o que as empresas podem mostrar aos seus clientes é a preocupação com a segurança e que aplica determinado processo de desenvolvimento com segurança.

A norma ISO/IEC 15408 surgiu com intuito de fornecer uma série de requisitos de segurança para que seja desenvolvido nos sistemas. Assim, pode-se medir o nível de segurança em que o sistema encontra-se, dando garantia para o cliente.

Uma tarefa difícil é avaliar se o software possui os requisitos descritos na norma ISO/IEC 15408. Por isso, o trabalho propôs implementar um software que ajude um auditor a fazer a avaliação dos requisitos de segurança conforme a norma. O auxilio está em diminuir as tarefas do auditor em procurar no sistema os requisitos implementados. No trabalho foram desenvolvidos três testes automatizados que testaram o sistema scrump, identificando se estava em conformidade com a norma. Assim, um sistema poderá ser avaliado e a empresa responsável poderá mostrar ao cliente que ele é mais seguro. Quando se investe em segurança investe-se também em qualidade de software.

Quanto aos trabalhos correlatos aqui apresentados, oferecem avaliação de outras normas de segurança, uma faz a analise da segurança de um servidor para indicar a adequação à norma ISO/IEC 27002 e outro apresenta um software para avaliação de uma empresa conforme a norma NBR ISO/IEC 17799 através de um *check list*. Sendo assim nenhum deles trabalha com a norma proposta. Já a ferramenta *Fortify* SCA é a que mais se enquadra nas características do trabalho, por fazer análise de um sistema e encontrar vulnerabilidades no sistema, porém, não relaciona os resultados obtidos com o requisito da norma ISO/IEC 15408.

4.1 EXTENSÕES

A seguir são apresentados alguns pontos que podem ser agregados ou melhorados no software. Segue as seguintes sugestões:

a) implementar testes para mais requisitos da norma ISO/IEC 15408 seguindo uma determinada regra;
b) fazer *plugins* com testes que possa ser possível testar outras linguagens que não seja Java;
c) melhorar o relatório para que utiliza uma ferramenta específica para geração de relatório;
d) implementar *plugin web service*.

REFERÊNCIAS BIBLIOGRÁFICAS

ALBUQUERQUE, Ricardo; RIBEIRO, Bruno. **Segurança no desenvolvimento de software**. Rio de Janeiro: Campus, 2002. 310 p.

ALLEN, Julia H. et al. **Software security engineering**: a guide for project managers. Upper Saddle River, NJ: Addison-Wesley, 2008. 334 p.

AZEVEDO, Denny. **Metodologias de segurança de sistemas**. São Paulo: IFSP, 2008. Disponível em: <http://gensys.eti.br/CEFET/Metodologias_Seguran%C3%A7a_Desenvolvimento_Sistemas.pdf>. Acesso em: 19 mar. 2009.

BATISTA, Carlos F. A. **Métricas de segurança de software**. 2007. 103 f. Dissertação (Mestrado em Informática) - Programa de Pós-Graduação em Informática, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro. Disponível em: <http://www.dominiopublico.gov.br/download/texto/cp040026.pdf>. Acesso em: 20 mar. 2009.

BRANDÃO, José E. M.S.; FRAGA, Joni da S**.** Gestão de riscos. In: SIMPÓSIO BRASILEIRO EM SEGURANÇA DA INFORMAÇÃO E DE SISTEMAS COMPUTACIONAIS, 8., 2008, Porto Alegre. **Anais...** Porto Alegre; SBC, 2008. p. 1-43.

BRAZ, Fabricio. **Software seguro**: O que esperar Fortify Souce Code Analyzer? Brasilia, 2008. Disponível em: <http://softwareseguro.blogspot.com/2008/03/o-que-esperar-fortify-souce-code.html>. Acesso em: 03 mar. 2010.

BURNETT, Steve; PAINE, Stephen. **Criptografia e segurança**: o guia oficial RSA. Rio de Janeiro: Elsevier: Campus, 2002. 367 p.

CODEHOUSE. **QDox**. [S.l.]: 2010. Disponível em: <http://qdox.codehaus.org/>. Acesso em: 10 abr. 2010.

COMMON CRITERIA. **The Common Criteria portal.** [S.l.][2009?]. Disponível em: <http://www.commoncriteriaportal.org/>. Acesso em: 21 mar. 2009.

CONALLEN, Jim**. Desenvolvendo aplicações WEB com UML**. Rio de Janeiro: Campus, 2003. 476 p.

CUGIK, Fernando L. S**oftware livre para verificação de adequação de servidores Gnu/Linux à norma de segurança NBR ISO/IEC 27002**. 2007. 74 f. Trabalho de Conclusão de Curso - (Bacharelado em Ciências da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

DIAS, Claudia. **Segurança e auditoria da tecnologia da informação**. Rio de Janeiro: Axcel Books, 2000. 218 p.

FRANZINI, Fernando. **Java para web**: autenticação e autorização em aplicativos web 2. [S.I.] 2009. Disponível em: <http://imasters.uol.com.br/artigo/14152/javaweb/autenticacao_e_autorizacao_em_aplicativos_web.html>. Acesso em: 04 maio. 2010.

GUERRA, Eduardo. Os sete pecados do controle de acesso em aplicações Java EE. **Mundojava**, Curitiba, n. 28, p. 26-36, abr. 2008.

HOWARD, Michael; LEBLANC, David. **Escrevendo código seguro:** estratégias e técnicas práticas para codificação segura de aplicativos em um mundo de rede. Porto Alegre: Bookman, 2005. 701 p.

ITEXTPDF. **your Java-PDF library**. [S.l.], 2010. Disponível em: <http://itextpdf.com/>. Acesso em: 04 abr. 2010.

LYRA, Maurício R. **Segurança e auditoria em sistemas de informação.** Rio de Janeiro: Ciência Moderna, 2008. 253 p.

NUNES, Francisco J.; BELCHIOR, Arnaldo D**. Processo seguro de desenvolvimento de software.**Ceará, 2006. Disponível em: <http://www.iadis.net/dl/final_uploads/200607C052.pdf>. Acesso em: 20 mar. 2009.

OLIVEIRA, Wilson J. **Segurança da informação:** técnicas e soluções. Florianópolis: Visual Books, 2001. 182 p.

ROSEMANN, Douglas. **Software para avaliação da segurança da informação de uma empresa conforme a norma NBR ISO-IEC 17799**. 2002. 93 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

SOURCEFORGE.NET. **Welcome to dom4j 2.0.** [S.l.], 2007. Disponível em: <http://dom4j.sourceforge.net/>. Acesso em: 21 mar. 2010.

THOMPSON, Marco A. **Proteção e segurança na internet**. São Paulo: Érica, 2002. 244 p.

ANEXO A – Divisão dos requisitos funcionais da norma 15.408

AUDITORIA DE SEGURANÇA
FAU_ARP - Resposta automática a auditoria
FAU_ARP.1 - Alarmes de segurança
FAU_GEN - Geração de dados para auditoria
FAU_GEN.1 - Geração de dados para auditoria
FAU_GEN.2 - Associação do usuário ao evento de auditoria
FAU_SAA - Análise da auditoria de segurança
FAU_SAA.1 - Análise de violação potencial
FAU_SAR - Revisão de dados da auditoria
FAU_SAR.1 - Revisão de auditoria
FAU_SEL - Auditoria seletiva
FAU_SEL.1 - Auditoria seletiva
FAU_STG - Armazenamento da trilha de auditoria
FAU_STG.1 - Armazenamento protegido da trilha de auditoria
FAU_STG.2 - Garantia da disponibilidade dos dados para auditoria

Quadro 15 - Classe auditoria de segurança

COMUNICAÇÃO
FCO_NRO - Não repúdio de origem
FCO_NRO.1 - Prova de origem
FCO_NRO.2 - Prova de origem assegurada
FCO_NRR - Não repúdio de recebimento
FCO_NRR.1 - Prova de recebimento seletiva
FCO_NRR.2 - Prova de recebimento assegurada

Quadro 16 – Classe comunicação

PROTEÇÃO DE DADOS DO USUÁRIO
FDP_ACF - Funções de segurança FDP_ACF.1 - Controle de acesso com base de atributos de segurança
FDP_ACC - Política de controle de acesso FDP_ACC.1 - Controle de acessos de subconjuntos
FDP_DAU - Autenticação de dados
FDP_DAU.1 - Autenticação básica dos dados FDP_DAU.2 - Autenticação dos dados com identidade do gerador
FDP_ETC - Exportação de dados para fora do controle do sistema FDP_ETC.1 - Exportação de dados sem atributos de segurança FDP_ETC.2 - Exportação de dados com segurança
FDP_IFC - Funções de controle de fluxo de informações FDP_IFC.1 - Controle de fluxo de informação
FDP_IFF - Política de controle de fluxo de informações FDP_IFF.1 - Atributos simples de segurança da informação
FDP_ITC - Importação de fora do controle das funções de segurança da aplicação FDP_ITC.1 - Importação de dados sem atributos de segurança FDP_ITC.2 - Importação de dados com segurança
FDP_ITT - Transferência interna FDP_ITT.1 - Proteção básica para transferência interna FDP_ITT.2 - Proteção baseada em atributo para transferencia interna FDP_ITT.3 - Monitoração de integridade
FDP_RIP - Proteção da informação residual FDP_RIP.1 - Proteção contra informação residual por subconjunto
FDP_ROL - Rollback (retorno) FDP_ROL.1 - Retorno básico
FDP_UCT - Confidencialidade de transferência de dados básica FDP_UCT.1 - Confidencialidade de transferência de dados básica
FDP_UIT - Proteção de integridade de dados do usuário FDP_UIT.1 - Integridade na transferência de dados

Quadro 17 – Classe Proteção de dados do usuário

IDENTIFICAÇÃO E AUTENTICAÇÃO
FIA_AFL - Falhas na autenticação
FIA_AFL.1 - Tratamento de falha de autenticação
FIA_ATD - Atributos do usuário para autenticação
FIA_ATD.1 - Definição de atributos do usuário para autenticação
FIA_SOS - Especificação de senhas
FIA_SOS.1 - Métrica mínima das senhas
FIA_SOS.2 - Capacidade de gerar senhas
FIA_UAU - Autenticação do usuário
FIA_UAU.1 - Ações anteriores à autenticação
FIA_UAU.2 - Autenticação do usuário antes de qualquer ação
FIA_UAU.3 - Autenticação à prova de fraude
FIA_UAU.4 - Autenticação de utilização única
FIA_UAU.5 - Múltiplos mecanismos de autenticação
FIA_UAU.6 - Re-autenticação
FIA_UAU.7 - Resposta restrita da autenticação
FIA_UID - Identificação do usuário
FIA_UID.1 - Ações anteriores à identificação
FIA_UID.2 - Identificação do usuário antes de qualquer ação
FIA_USB - Ligação do usuário com o sistema
FIA_USB.1 - Ligação do usuário com o sistema

Quadro 18 – Identificação e autenticação

GERENCIAMENTO DE SEGURANÇA
FMT_MOF - Gerenciamento de funções de segurança
FMT_MOF.1 - Gerenciamento de funções de segurança
FMT_MSA - Gerenciamento de atributos de segurança
FMT_MSA.1 - Gerenciamento de atributos de segurança
FMT_MSA.2 - Segurança de atributos de segurança
FMT_MSA.3 - Inicialização de atributos estáticos
FMT_MTD - Gerenciamento de dados de segurança
FMT_MTD.1 - Gerenciamento de dados de segurança
FMT_MTD.2 - Gerenciamento dos limites de dados de segurança
FMT_SMF - Especificação do gerenciamento de funções
FMT_SMF.1 - Especificação do gerenciamento de funções
FMT_SMR - Papéis de gerenciamento de segurança
FMT_SMR.1 - Papéis de segurança
FMT_SMR.2 - Restrição nos papéis de segurança
FMT_SMR.3 - Incorporação de papéis de segurança

Quadro 19 – Classe Gerenciamento de segurança

AUTOPROTEÇÃO
FPT_AMT - Teste da camada subjacente
FPT_AMT.1 - Teste da camada subjacente
FPT_ITA - Disponibilidade de dados exportados pela aplicação
FPT_ITA.1 - Disponibilidade de dados exportados pela aplicação
FPT_ITC - Confidencialidade dos dados exportados pela aplicação
FPT_ITC.1 - Confidencialidade dos dados exportados pela aplicação
FPT_ITI - Integridade dos dados exportados pela aplicação
FPT_ITI.1 - Detecção de modificações
FPT_ITT - Proteção na transferência interna de dados
FPT_ITT.1 - Proteção básica de dados internos da aplicação
FPT_ITT.2 - Separação de dados de segurança
FPT_ITT.3 - Monitoração de integridade de dados
FPT_PHP - Proteção física do sistema
FPT_PHP.3 - Resistência a ataque físico
FPT_RPL - Detecção de repetição
FPT_RPL.1 - Detecção de repetição
FPT_RVM - Monitor de referência
FPT_RVM.1 - Não-contorno da política de segurança
FPT_SSP - Protocolo de sincronismo de estado
FPT_SSP.1 - Protocolo de sincronismo simples
FPT_SSP.2 - Protocolo de sincronismo mútuo
FPT_STM - Registro de tempo
FPT_STM.1 - Registros de tempo
FPT_TDC - Consistência de dados entre funções de segurança
FPT_TDC.1 - Consistência de dados entre funções de segurança
FPT_TRC - Consistência de dados replicados
FPT_TRC.1 - Consistência de dados replicados
FPT_TST - Autoteste
FPT_TST.1 - Autoteste

Quadro 20 – Classe Autoproteção

UTILIZAÇÃO DE RECURSOS
FRU_PRS - Prioridade de serviços
FRU_PRS.1 - Priorização de serviços limitada
FRU_RSA - Alocação de recursos
FRU_RSA.1 - Cota máxima de utilização

Quadro 21 –Classe Utilização de recursos

ACESSO AO SISTEMA
FTA_LSA - Limitação de escopo ao sistema
FTA_LSA.1 - Limitação de escopo no acesso ao sistema
FTA_MCS - Limitação do número de seções simultâneas
FTA_MCS.1 - Limitação básica do numero de seções
FTA_MCS.2 - Limitação básica do numero de sessões por usuário
FTA_SSL - Travamento de sessão
FTA_SSL.1 - Travamento automático de sessão
FTA_SSL.2 - Travamento de sessão por requisição do usuário
FTA_SSL.3 - Encerramento automático da sessão
FTA_TAB - Mensagem de acesso
FTA_TAB.1 - Mensagem de acesso
FTA_TAH - Histórico de acesso
FTA_TAH.1 - Histórico de acesso
FTA_TSE - Limitação de acesso ao sistema
FTA_TSE.1 - Limitação de acesso ao sistema

Quadro 22 – Classe Acesso ao sistema

CAMINHOS OU CANAIS CONFIÁVEIS
FTP_ITC - Canal confiável entre funções de segurança
FTP_ITC.1 - Canal confiável entre funções de segurança
FTP_TRP - Caminho confiável
FTP_TRP.1 - Caminho confiável

Quadro 23 – Classe Caminhos ou canais confiáveis