UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO

PLUGINS PARA TESTES SEMI - AUTOMATIZADOS DE CONFORMIDADE COM A NORMA ISO/IEC 15408

DIONEI HERKENHOFF

BLUMENAU
2011

2011/1-15

DIONEI HERKENHOFF

PLUGINS PARA TESTES SEMI-AUTOMATIZADOS DE CONFORMIDADE COM A 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, Mestre – Orientador

BLUMENAU
2011

2011/1-15

PLUGINS PARA TESTES SEMI-AUTOMATIZADOS DE CONFORMIDADE COM A NORMA ISO/IEC 15408

Por

DIONEI HERKENHOFF

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, 27 de junho de 2011

Dedico este trabalho aos meus pais, minha noiva e meus irmãos que sempre me incentivaram a estudar.

AGRADECIMENTOS

Aos meus pais, meus irmãos e minha noiva que sempre estiveram ao meu lado, me incentivando nos estudos e na vida.

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.

“Não há problema que não possa ser solucionado pela paciência”.

Chico Chavier.

RESUMO

Este trabalho apresenta o desenvolvimento de testes semi-automatizados que serão acoplados ao Software para verificação de conformidade de segurança com base na norma ISO/IEC 15408, também chamado de SCAN-CC, para que sejam realizados testes semi-automatizados, verificando se o sistema analisado atende as regras de segurança definidas na norma.

Palavras-chave: Segurança.Testes.Semi-Automatizado.ISO/IEC 15408.

ABSTRACT

This paper presents the development of semi-automated tests that will be attached to the Software for compliance verification of security based on ISO / IEC 15408, also called SCAN-DC, that tests be performed semi-automated checking if the systemanalysis meets the security rules defined in the standard.

Key-words: Security.Testing.Half-Automated.ISO/IEC 15408.

LISTA DE ILUSTRAÇÕES

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*

ISO – *International Standardization Organization*

ITSEC – *Information Technology Security Evaluation Criteria*

ST – *Security Target*

TCSEC – *Trusted Computer Security Evaluation Criteria*

TOE – *Target Of Evaluation*

UC – Caso de uso

UML – *Unified Modeling Language*

XML – *eXtensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

A segurança em sistema é um requisito tão básico quanto velocidade: o cliente quer, mesmo que não diga isso explicitamente. O problema com relação à segurança em um sistema, muitas vezes, só se torna uma preocupação quando este ocorre e, mesmo que não tenha havido prejuízo por parte do cliente, a confiança dele com relação ao sistema já fica comprometida (ALBUQUERQUE; RIBEIRO, 2002, p. 1).

O assunto normalmente é abordado quando se trata de sistemas para a internet, que são mais suscetíveis a algum tipo de ataque, pois estão disponíveis para todos, facilitando o acesso de pessoas mal intencionadas (ALBUQUERQUE; RIBEIRO, 2002, p. 1). Mas esta preocupação não pode ser direcionada apenas a sistemas da internet, pois, ao contrário do que parece, a segurança não se trata apenas de garantir que informações sigilosas caiam em mãos erradas, mas também garantir que a informação passada seja a real, ou seja, que não foi alterada por pessoas sem autorização e garantir que o sistema realize as suas tarefas sem falhas, por exemplo.

Com a integração dos sistemas, onde as empresas têm centralizado a administração de suas filiais e utilizam-se da internet como forma de transmissão de dados entre matriz e filial, a segurança deve ter um papel fundamental no processo de planejamento e implementação do sistema. Segundo Albuquerque e Ribeiro (2002, p. 1-2), os principais desafios para uma equipe de desenvolvimento com relação à segurança, são como desenvolver uma aplicação segura e como garantir esta segurança.

Para atender a necessidade de criação de um padrão para o desenvolvimento seguro de um sistema e evitar que fossem criados diversos padrões divergentes, foi instituído o comitê internacional *Common Criteria* (CC) e em 1999 foi homologada a norma ISO/IEC 15408 que define classes com famílias de atributos de segurança que o sistema deve atender para caracterizar-se um sistema seguro (ALBUQUERQUE; RIBEIRO, 2002, p. 7-8).

Diante do exposto, torna-se evidente a crescente preocupação com segurança e a necessidade do desenvolvimento de ferramentas que possam realizar verificações de requisitos de segurança de forma automatizada, dando agilidade ao processo no desenvolvimento e manutenção do sistema.

Software para Verificação de Conformidades de Sistemas à Norma ISO/IEC 15408 (TRAPP, 2010, p. 1), que a partir deste momento será referenciado como SCAN-CC, é uma ferramenta que dá suporte a realização e criação de um plano de auditoria e realiza a

verificação de conformidades de segurança através de testes que podem ser desenvolvidos e agregados a ferramenta.

Este trabalho visa então desenvolver *plugins* de testes semi-automatizados para duas famílias de atributos de segurança da norma de segurança ISO/IEC 15408, a família *security AUdit* (FAU), *Identification and Authentication* (FIA), adaptando-os ao SCAN-CC.

1.1 OBJETIVOS DO TRABALHO

O objetivo deste trabalho é desenvolver *plugins* de testes para verificação de conformidades a norma internacional de segurança, ISO/IEC 15408. Os objetivos específicos do trabalho são:

a) realizar testes semi-automatizados, em sistemas desenvolvidos em Java, verificando as regras de geração de dados para auditoria e auditoria selecionável, , da classe de auditoria e segurança;
b) realizar testes semi-automatizados em sistemas desenvolvidos em Java, verificando as regras de análise de violação em potencial, revisão da trilha de auditoria e auditoria de resposta automática, da classe de auditoria e segurança;
c) realizar testes semi-automatizados em sistemas desenvolvidos em Java, verificando as regras de tratamento a falhas de autenticação, definição de atributos do usuário e especificação de senhas, da classe de identificação e autenticação.

1.2 ESTRUTURA DO TRABALHO

Além deste, o trabalho está organizado em três capítulos, intitulado 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, norma ISO/IEC 15408, SCAN-CC: Software para verificação de conformidade de sistemas à norma ISO/IEC 15408 e por fim são apresentados os trabalhos correlatos.

O capítulo 3 é dedicado à especificação do trabalho, os principais casos de uso e

também 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 e sua importância. Em seguida, é apresentada a norma de segurança ISO/IEC 15408 para o desenvolvimento de software. Posteriormente, o protótipo de verificação de conformidades de segurança de sistemas com base na norma ISO/IEC 15408, SCAN-CC. Por fim, são apresentados os trabalhos correlatos.

2.1 SEGURANÇA DA INFORMAÇÃO

Conforme Campos (2006, p. 6), um sistema de segurança baseia-se em três princípios básicos:

a) confidencialidade: o princípio da confidencialidade é respeitado quando apenas pessoas explicitamente autorizadas podem ter acesso à informação;
b) integridade: a integridade é respeitada quando a informação acessada está completa, sem alterações e, portanto, confiável;
c) disponibilidade: a disponibilidade é respeitada quando a informação está acessível, para pessoas autorizadas, sempre que necessário.

Além destes três aspectos principais, Lyra (2008, p. 4) descreve aspectos como autenticação, não-repúdio, legalidade, privacidade e auditoria para segurança da informação.

Dias (2000, p. 44) comenta que, embora todos os aspectos sejam importantes para a segurança da informação, eles devem ser levados em conta de acordo com necessidade. Um sistema que não contém dados confidenciais, mas necessita estar disponível 24 horas por dia, não requer privacidade, mas sim disponibilidade.

Para Albuquerque e Ribeiro (2002, p. 32), embora possa parecer um processo trabalhoso, é preciso fazer um levantamento das ameaças que o sistema possa sofrer e montar uma tabela de ameaças e enquadrar os requisitos de segurança do sistema. Albuquerque e Ribeiro (2002, p. 32) ainda citam que existem três preocupações básicas, que devem ser levadas em conta, com relação à segurança em desenvolvimento de um software:

a) segurança do ambiente de desenvolvimento: deve se ter a preocupação em manter os códigos fonte seguros. Evitar o roubo de código ou indisponibilidade da equipe

de desenvolvimento;

b) segurança da aplicação desenvolvida: deve-se seguir corretamente a especificação de segurança;
c) garantia de segurança da aplicação desenvolvida: é preciso garantir a segurança da aplicação em desenvolvimento.

Mesmo que um sistema atenda a todos os itens mencionados anteriormente, de acordo com Albuquerque e Ribeiro (2002, p. 4) "é impossível prevenir todos os ataques [...]. O sistema seguro é aquele que concentra suas defesas nos ataques mais prováveis e com maior perda".

2.2 NORMA ISO/IEC 15408

A norma ISO/IEC 15408 é um padrão criado a partir da união dos padrões *Trusted Computer Security Evaluation Criteria* (TCSEC), *Information Technology Security Evaluation Criteria* (ITSEC), *Canadian Trusted Computer Product Evaluation* (CTCPE) e o *Federal Criteria* (FC), para evitar que fossem criados diversos padrões divergentes.

O CC estabelece que qualquer sistema, para ser considerado seguro, precisa ter o seu *Security Target*[1] (objetivo ou alvo de segurança) elaborado e emprega também o termo *Target Of Evaluation* (TOE) para referenciar o sistema que está sendo desenvolvido ou avaliado (ALBUQUERQUE; RIBEIRO, 2002, p. 8).

O CC define sete níveis de garantia de segurança denominados de *Evaluation Assurance Level* (EAL). A cada nível há um maior número de testes e, portanto, maior garantia de que o sistema atende aos requisitos de segurança. "Na ISO/IEC 15408, os níveis EAL-5 a EAL-7 foram considerados extremamente rígidos, na prática, inviáveis" (ALBUQUERQUE; RIBEIRO, 2002, p. 8).

Além das regras contidas nos níveis de garantia de segurança definidos pela CC, existem ainda atributos com regras para realização de testes para verificação de conformidades de segurança, estes agrupados em famílias de acordo com suas características. Estes requisitos de segurança definidos não se destinam a ser uma resposta definitiva para os

[1] *Security Target* é a especificação de segurança, ou seja, indica quais aspectos de segurança foram considerados importantes e porque o foram para aquele sistema em particular (ALBUQUERQUE; RIBEIRO, 2002, p. 8)

problemas de segurança de informação, mas como conjunto de regras de segurança que possam ser usados para criar produtos confiáveis, refletindo as necessidades do mercado (COMMOM CRITERIA, 2009, p. 17).

As famílias mencionadas anteriormente são nomeadas como *Family security AUdit* (FAU), *Family Identification and Authentication* (FIA), *Family TOE Access* (FTA), *Family Cryptografic Support* (FCS), *Family Communication* (FCO), *Family Trusted Paths/channels* (FTP), *Family Protection of the TOE security functions* (FPT), *Family Privacy* (FPR), *Family user Data Protection* (FDP), *Family security ManagemenT* (FMT) e a *Family Resorce Utilisation* (FRU) (CYGNACON, 2011). De acordo Albuquerque e Ribeiro (2002, p. 114-238), cada família possui a seguinte característica:

a) FAU – família com atributos de segurança com regras para registro e visualização da trilha de auditoria do sistema. Este será descrito de forma mais detalhada do decorrer deste trabalho;
b) FIA – família com atributos de segurança com regras para o tratamento da identificação e autenticação do usuário ao sistema. Este será descrito de forma mais detalhada do decorrer deste trabalho;
c) FTA – família com atributos de segurança com regras para definir o controle de acesso do usuário ao sistema;
d) FCS – família com atributos de segurança com regras para especificação e uso de criptografia no sistema;
e) FCO – família com atributos de segurança com regras para o tratamento de ataques de repúdio[2] ao sistema;
f) FTP – família com atributos de segurança com regras para especificação e uso de canal seguro ou confiável de comunicação entre o sistema e o usuário ou entre sistemas e outros sistemas;
g) FPT – família com atributos de segurança com regras para especificação da proteção de dados de segurança do sistema;
h) FPR – família com atributos de segurança com regras para manter a privacidade[3] dos dados do usuário;

---

[2] Repúdio é uma forma de ataque onde o agente do ataque executa uma função no sistema e posteriormente nega tê-la efetuado (ALBUQUERQUE; RIBEIRO, 2002, p. 167).

[3] Privacidade é a capacidade de um usuário realizar ações em um sistema sem que seja identificado (ALBUQUERQUE; RIBEIRO, 2002, p. 206).

i) FDP – família com atributos de segurança com regras para especificação da proteção de dados do usuário do sistema;
j) FMT – família com atributos de segurança com regras para o gerenciamento de segurança do sistema;
k) FRU – família com atributos de segurança com regras de segurança no uso de recursos do sistema.

2.2.1 Família FAU - Registro e visualização de dados da auditoria

Auditoria de segurança envolve o reconhecimento, gravação, armazenamento e análise de informações relacionadas às atividades relevantes a segurança (COMMOM CRITERIA, 2009, p.29). Auditoria em software significa uma parte da aplicação, ou um conjunto de funções do sistema, que viabiliza uma auditoria. Isso ocorre pela gravação e manutenção de uma trilha de ações realizadas no sistema e, posteriormente, pela análise ou visualização desta, ou seja, o sistema mantém os registros de tudo o que foi feito nele de forma que, em caso de problema de segurança, alguém possa identificar o que ou quem o causou (ALBUQUERQUE E RIBEIRO, 2002, P. 114).

Para o desenvolvimento do registro e visualização de dados de auditoria no TOE, o CC define atributos com regras que devem ser seguidas para o atendimento destas regras. De acordo com Albuquerque e Ribeiro (2002, p. 114-128), estes atributos são `FAU_GEN`, `FAU_SEL`, `FAU_SAR`, `FAU_SAA`, `FAU_ARP` e `FAU_STG`, cada um com as seguintes características:

a) `FAU_GEN` – tratam das características que as funções de segurança do TOE devem apresentar. Estas regras definem que as funções devem ser capazes de gerar um registro de auditoria para cada evento pré-determinado, como na inicialização e finalização das funções de auditoria, por exemplo. Define também quais informações devem ser armazenadas a cada evento de auditoria, como a data e hora, o tipo de evento, identidade do usuário ou sistema entre outros e que devem ser capazes de associar cada evento auditado ao usuário ou sistema que o gerou;
b) `FAU_SEL` – define que o TOE deve permitir ou não a seleção dos eventos que irão gerar dados para auditoria, ou seja, que possa escolher um subconjunto de eventos onde seja registrado somente quando houver o acesso ao sistema, por exemplo;
c) `FAU_SAR` – determina que as funções de segurança do TOE devem permitir que

apenas determinados usuários leiam a trilha de auditoria e que estes registros devam ser apresentados de forma que o usuário com permissão possa ler e interpretá-los. Verifica também as informações da auditoria apresentadas pela função de segurança do TOE, deve também permitir que o usuário que esta revisando, deve ter a possibilidade de realizar buscas e ordenações, pela data, hora ou tipo de evento, por exemplo, para facilitar a busca por determinados eventos;

d) FAU_SAA – apresenta regras definindo requisitos para a análise de violação em potencial, onde as funções de segurança do TOE devem ser capazes de aplicar um conjunto de regras: monitoração dos eventos auditados, a detecção de anomalia baseada em perfil de uso, a detecção de ataques por heurística simples, eventos que podem determinar uma invasão, a seqüência e cenários;
e) FAU_ARP - auditoria com resposta automática. Este atributo possui apenas uma regra que define a forma com que o TOE deve se comportar quando da detecção de uma invasão. Esta regra define que o TOE deve executar uma determinada lista de ações no momento em que for detectada uma possível invasão;
f) FAU_STG – Esta regra define que a trilha de auditoria deve ser armazenada de forma protegida, não permitindo sua remoção e/ou alteração por usuários não autorizados, garantir o armazenamento mínimo de dados em caso de falha, quais ações devem ser tomadas no momento em que estiver terminando o espaço para a trilha de auditoria, a prevenção contra a perda dos dados e quais ações realizar quando não há mais espaço para o armazenamento da trilha, evitando a perda de informações.

2.2.2 Família FIA – Identificação e Autenticação

Identificação e autenticação podem parecer ter o mesmo conceito, mas a "identificação do usuário trata de saber quem está operando o sistema. Autenticação do usuário trata de garantir que o usuário é quem ele diz ser" (ALBUQUERQUE; RIBEIRO, 2002, p. 130). Ainda segundo Albuquerque e Ribeiro (2002, p. 129-154), FIA_UID, FIA_UAU, FIA_AFL, FIA_ATD, FIA_SOS e FIA_USB são atributos que definem regras para o desenvolvimento seguro, com o foco na identificação e autenticação do usuário. Estas regras são descritas da seguinte forma:

a) FIA_UID – segundo a CC, as funções de segurança do TOE podem identificar

quais ações o usuário poderá realizar no sistema sem ser identificado. Para executar outras funções não definidas, o usuário deverá ser identificado;

b) FIA_UAU – segundo a CC, as funções de segurança do TOE podem identificar quais ações o usuário poderá realizar no sistema sem ser autenticado, define que as funções de segurança do TOE devem impedir/detectar a tentativa de uso de dados de autenticação que tenham sido forjados;

c) FIA_AFL – atributo que define regras para o tratamento de falha de autenticação. As regras deste atributo definem que as funções de segurança do TOE deve detectar quando há uma quantidade de tentativas falhas de autenticação, caracterizando uma tentativa de invasão por tentativa e erro. As funções do TOE também devem definir ações a serem tomadas após a ocorrência da tentativa de invasão, descrita anteriormente;

d) FIA_ATD – segundo este atributo, as funções de segurança devem manter uma lista de atributos de segurança para cada usuário, como identificação do usuário, prazo de validade dos dados da autenticação, entre outros;

e) FIA_SOS – as funções de segurança, de acordo com este atributo, devem fornecer um mecanismo garanta que a senha atenda a determinada métrica, fornecer um mecanismo que possa gerar senhas que atendam a esta métrica e devem garantir o uso das senhas geradas para determinadas funções de segurança da aplicação;

f) FIA_USB – atributo que define que as funções de segurança da aplicação devem fornecer um mecanismo que associe atributos de segurança do usuário a processos executados por ele, como por exemplo, conexões simultâneas com o mesmo usuário, limitação de acesso entre outros.

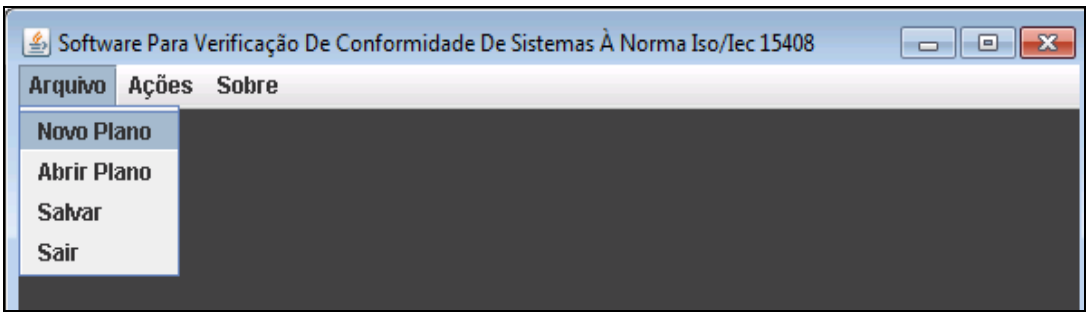
2.3 SCAN-CC – SOFTWARE DE VERIFICAÇÃO DE CONFORMIDADES DE SEGURANÇA À NORMA ISO/IEC 15408

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).

Com base no que foi descrito anteriormente, o SCAN - CC nome dado ao Software

para verificação de conformidades de sistemas à norma ISO/IEC 15408 (TRAPP, 2010, p. 1), foi desenvolvido com objetivo de dar suporte à realização de auditorias em sistemas e execução de testes para verificação de conformidades de segurança com base na norma ISO/IEC 15408.

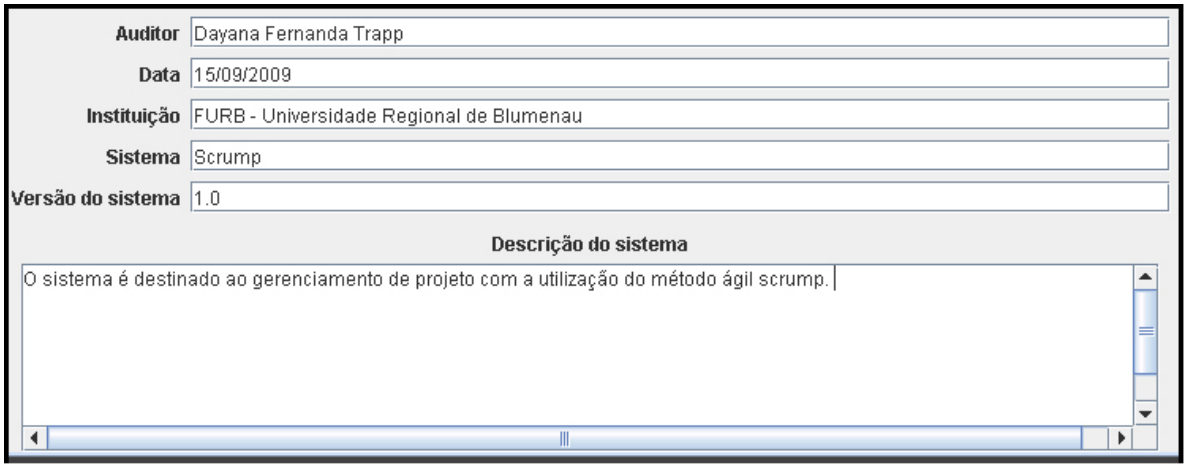
Para realização de um plano de auditoria, o auditor cria um novo plano, conforme Figura 1, inserindo dados e selecionando quais requisitos de segurança que serão avaliados durante o processo, Figura 2. Todos os requisitos da norma ISO/IEC 15408 são persistidos em um arquivo XML que é lido a cada novo plano de auditoria. Após a seleção dos requisitos a serem verificados, Figura 3, o auditor realiza a execução dos testes de verificação. Após a execução é apresentada uma tela onde são apresentados todos os requisitos que devem ser avaliados. O auditor percorrerá todos os requisitos verificando quais testes foram realizados de forma automatizada e os que não foram verificados, o auditor tem como classificar manualmente se o requisito está ou não sendo atendido. Por fim, pode-se gerar um relatório com os dados da auditoria ou salvá-lo para recuperação futura.



Fonte: (TRAPP, 2010, p. 50)

Figura 1 – Criação de um novo plano de auditoria

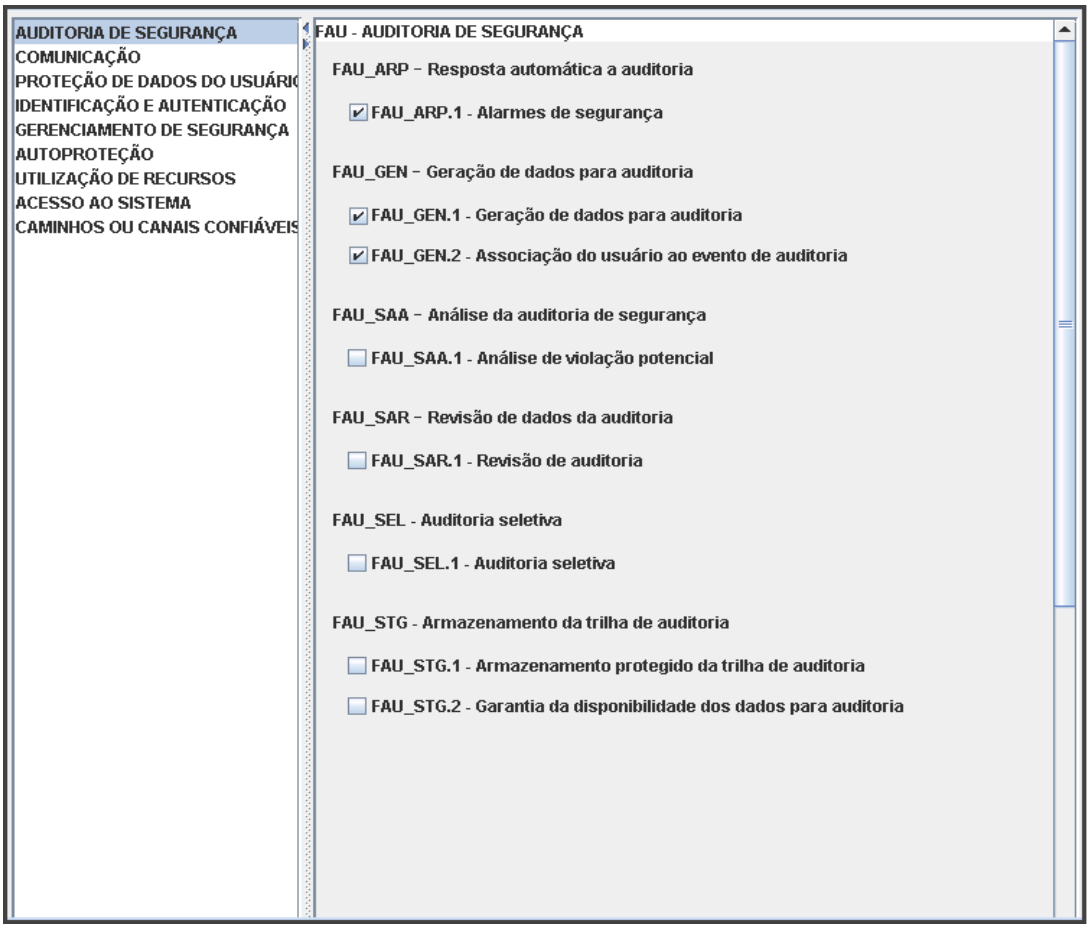
Segundo Trapp (2010, p. 45), o software somente irá executar uma auditoria automática quando forem inseridos *plugins* com a implementação dos testes automatizados. Para os testes que não possuírem *plugins* desenvolvidos, a verificação deve ser realizada manualmente. Para que eles sejam inseridos na ferramenta, é disponibilizada uma *Application Programming Interface* (API) para que o auditor possa implementar *plugins* de testes automatizados para cada requisito da norma.



Fonte: Trapp (2010, p. 50).

Figura 2 - Cadastro básico do sistema auditado

O SCAN-CC possui poucos testes implementados, logo há uma grande necessidade de desenvolvimento destes, de forma a realizarem verificações semi-automatizadas e aperfeiçoar o processo de realização da auditoria.



Fonte: Trapp (2010, p. 50).

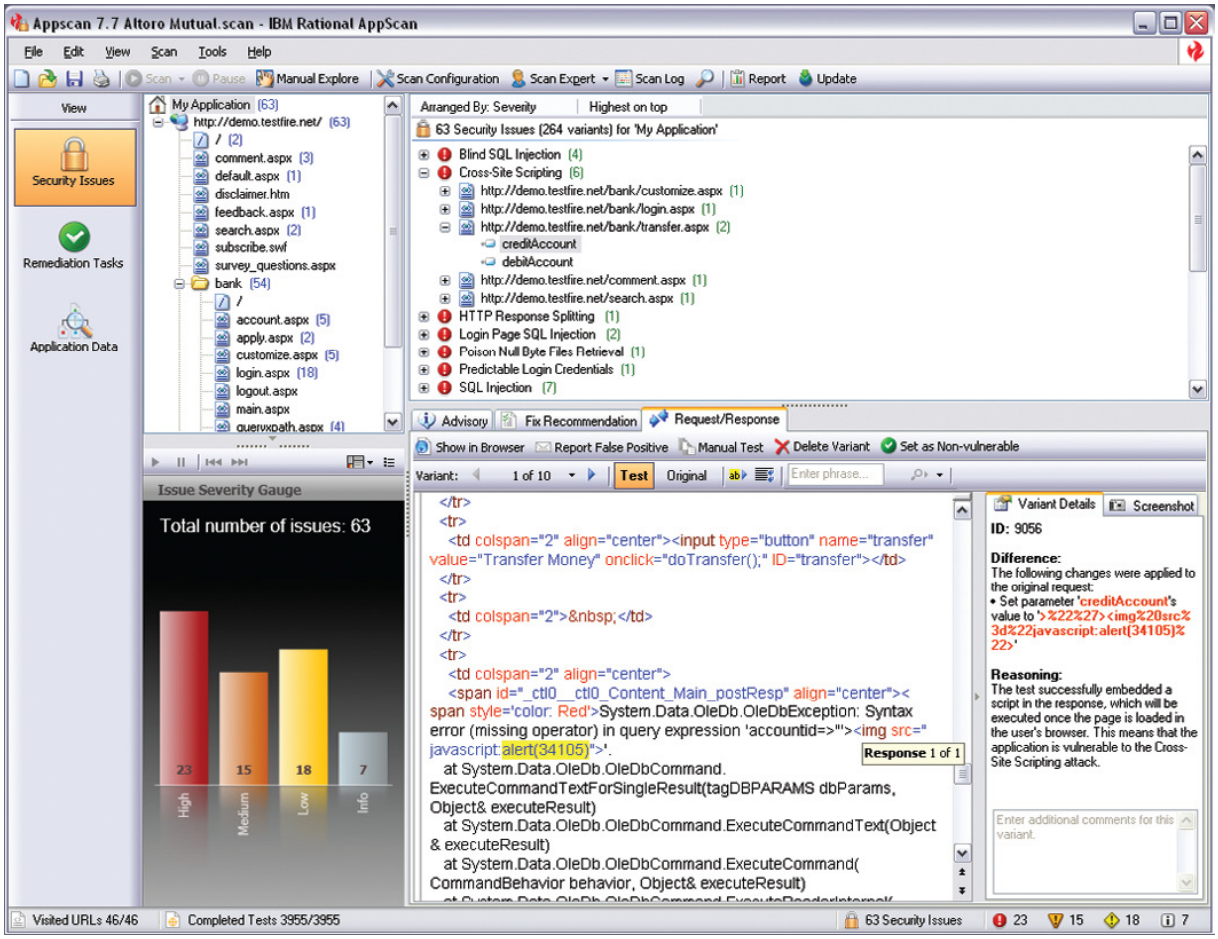
Figura 3 - Seleção dos requisitos a serem auditados

2.4 TRABALHOS CORRELATOS

Há uma série de trabalhos semelhantes ao proposto e dentre eles estão o *IBM Rational® AppScan® Express Edition e o Lapse.*

IBM Rational® AppScan® Express Edition é uma ferramenta comercial, desenvolvida pela IBM, para verificação de vulnerabilidades de segurança para sistemas da *web*. Suas principais funcionalidades estão relacionadas à realização de testes automatizados para verificação de vulnerabilidades em aplicações da *web*. Segundo a IBM (2010) a ferramenta varre automaticamente aplicativos *web* usando tecnologias da *web* 2.0 como Adobe Flash, *JavaScript* entre outros, reduz drasticamente a necessidade de teste manual e gera relatórios de conformidade com base em normas de segurança como ISO 17799 e ISO 27001. NaFonte: IBM (2010).

Figura 4 e Figura 5 pode se verificar a ferramenta realizando a análise do sistema e a exibição de um aviso de segurança.



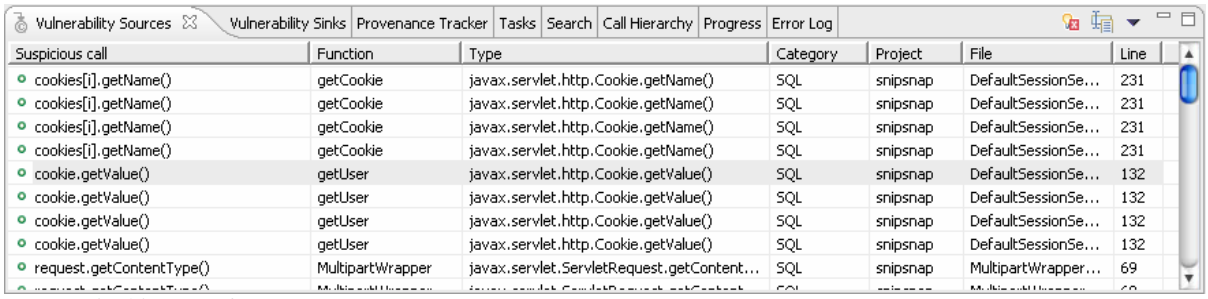
Fonte: IBM (2010).

Figura 4 – Visualização dos problemas de segurança.

Fonte: IBM (2010).

Figura 5 – Visualização do aviso de segurança da ferramenta.

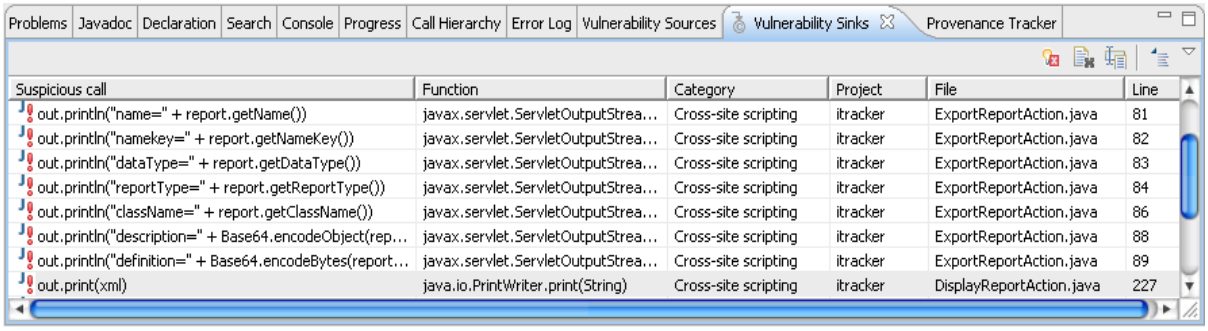
Lapse é uma ferramenta sob licença *General Public License* (GPL), projetada para o auxílio na tarefa de realização de auditorias e testes de segurança em aplicações Java J2EE para tipos comuns de vulnerabilidades encontradas em aplicações *web* (LIVSHITS, 2006). Trata se de uma ferramenta para realização de testes de verificação de segurança, utilizando a técnica de análise estática do código fonte da aplicação. Para realização dos testes, é necessário que o projeto do *Lapse* e o projeto que será testado estejam abertos na ferramenta para desenvolvimento JAVA, *Eclipse.* Após ser executado, o *Lapse* informa alguns dados sobre os erros encontrados, como o tipo da vulnerabilidade e o local onde ocorre o problema, conforme Figura 6 e Figura 7. O *Lapse* verifica vulnerabilidades como c*ross-site scripting*, *SQL injections* entre outras inconsistências de segurança encontradas em aplicações web.



Suspicious call	Function	Type	Category	Project	File	Line
cookies[i].getName()	getCookie	javax.servlet.http.Cookie.getName()	SQL	snipsnap	DefaultSessionSe...	231
cookies[i].getName()	getCookie	javax.servlet.http.Cookie.getName()	SQL	snipsnap	DefaultSessionSe...	231
cookies[i].getName()	getCookie	javax.servlet.http.Cookie.getName()	SQL	snipsnap	DefaultSessionSe...	231
cookies[i].getName()	getCookie	javax.servlet.http.Cookie.getName()	SQL	snipsnap	DefaultSessionSe...	231
cookie.getValue()	getUser	javax.servlet.http.Cookie.getValue()	SQL	snipsnap	DefaultSessionSe...	132
cookie.getValue()	getUser	javax.servlet.http.Cookie.getValue()	SQL	snipsnap	DefaultSessionSe...	132
cookie.getValue()	getUser	javax.servlet.http.Cookie.getValue()	SQL	snipsnap	DefaultSessionSe...	132
cookie.getValue()	getUser	javax.servlet.http.Cookie.getValue()	SQL	snipsnap	DefaultSessionSe...	132
request.getContentType()	MultipartWrapper	javax.servlet.ServletRequest.getContent...	SQL	snipsnap	MultipartWrapper...	69

Fonte: Livshits (2006).

Figura 6 – Verificação de chamadas de métodos suspeitos



Suspicious call	Function	Category	Project	File	Line
out.println("name=" + report.getName())	javax.servlet.ServletOutputStrea...	Cross-site scripting	itracker	ExportReportAction.java	81
out.println("namekey=" + report.getNameKey())	javax.servlet.ServletOutputStrea...	Cross-site scripting	itracker	ExportReportAction.java	82
out.println("dataType=" + report.getDataType())	javax.servlet.ServletOutputStrea...	Cross-site scripting	itracker	ExportReportAction.java	83
out.println("reportType=" + report.getReportType())	javax.servlet.ServletOutputStrea...	Cross-site scripting	itracker	ExportReportAction.java	84
out.println("className=" + report.getClassName())	javax.servlet.ServletOutputStrea...	Cross-site scripting	itracker	ExportReportAction.java	86
out.println("description=" + Base64.encodeObject(rep...	javax.servlet.ServletOutputStrea...	Cross-site scripting	itracker	ExportReportAction.java	88
out.println("definition=" + Base64.encodeBytes(report...	javax.servlet.ServletOutputStrea...	Cross-site scripting	itracker	ExportReportAction.java	89
out.print(xml)	java.io.PrintWriter.print(String)	Cross-site scripting	itracker	DisplayReportAction.java	227

Fonte: Livshits (2006).

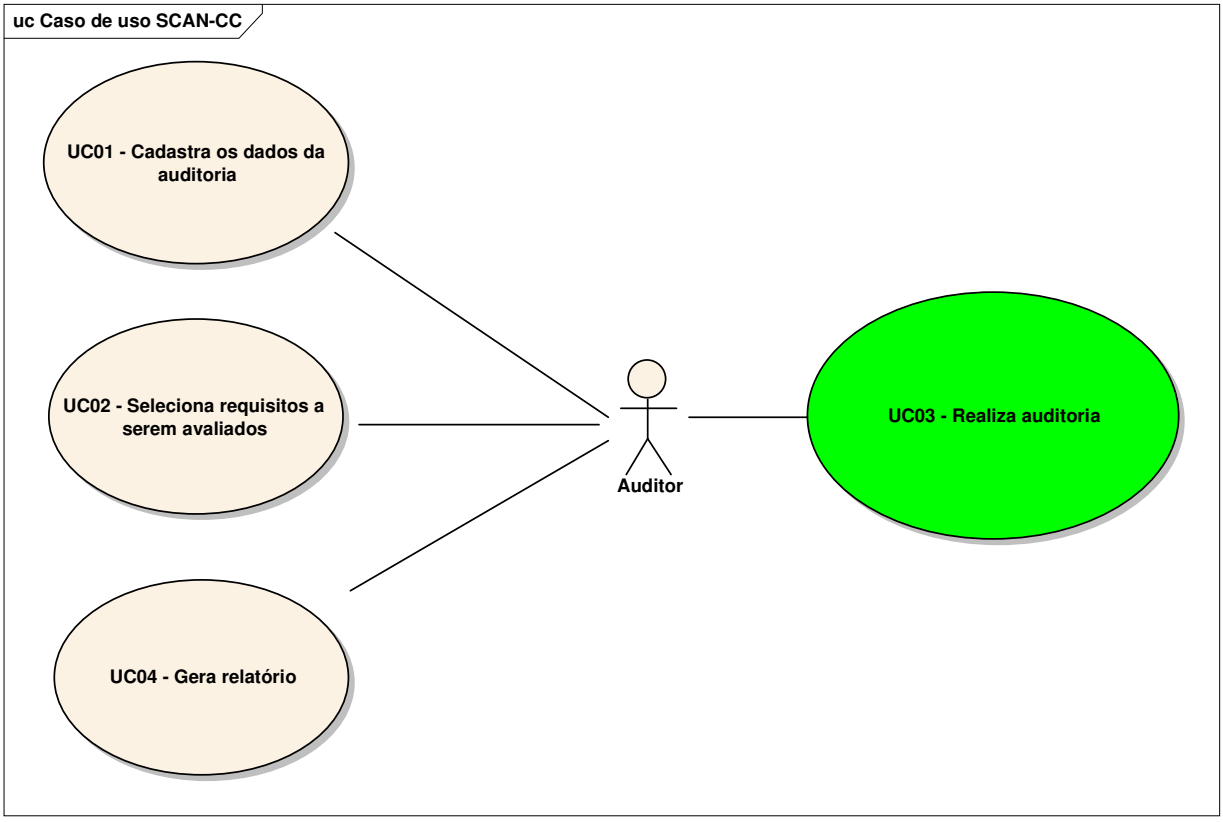
Figura 7 – Tela em que *Lapse* mostra os erros encontrados

3 DESENVOLVIMENTO

Este capítulo contém análise do protótipo, SCAN-CC, os requisitos, a especificação, diagrama de estados e diagramas de classe do software desenvolvido e a operacionalidade do sistema.

3.1 ANÁLISE DO PROTÓTIPO SCAN-CC

Como mencionado, o SCAN-CC é uma ferramenta para realização de auditorias em softwares verificando conformidades com a norma ISO/IEC 15408. Na Figura 8 são apresentados os Casos de Uso (UC) do SCAN-CC, representando as principais funcionalidades do sistema.



Fonte: adapt. Trapp(2010, p. 31 e 32).

Figura 8 – Diagrama de Casos de uso do protótipo SCAN-CC

O UC destacado na Figura 8, é o caso que descreve o processo de realização dos testes semi-automatizados da ferramenta, conforme Quadro 1, onde serão realizadas alterações com

a inserção de novos testes semi-automátizados.

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.

Fonte: Trapp (2010, p. 32).

Quadro 1 - Caso de uso `Realiza auditoria automática`

A especificação das alterações que foram realizadas vão ser exibidas nos próximos capítulos.

3.2 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

Segue os principais requisitos do software, baseado nos requisitos de segurança da norma ISO/IEC 15408:

b) o sistema deve solicitar um arquivo do tipo *eXtensible Markup Language* (XML) com o endereço completo do diretório do sistema, os pacotes da aplicação e o endereço do arquivo de *classpath* do sistema a ser verificado (Requisito Funcional - RF);

c) o sistema deve realizar a análise automática dos requisitos de auditoria de segurança de acordo com os requisitos selecionados pelo auditor e com a norma (RF);

d) o sistema deve realizar a análise automática dos requisitos de identificação e autenticação de acordo com os requisitos selecionados pelo auditor e com a norma

(RF);

e) o sistema deve realizar a auditoria em sistemas desenvolvidos na linguagem JAVA (RF);
f) o sistema deve ser desenvolvido usando orientação a objeto (Requisito Não Funcional – RNF);
g) o sistema deve ser implementado na linguagem de programação Java (RNF).

3.3 ESPECIFICAÇÃO

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

3.3.1 Diagrama de Estados

Na Figura 9 é apresentado o diagrama de estados, exemplificando os passos que ocorrem durante a execução dos testes semi-automatizados, que são verificados no UC destacado na Figura 8.

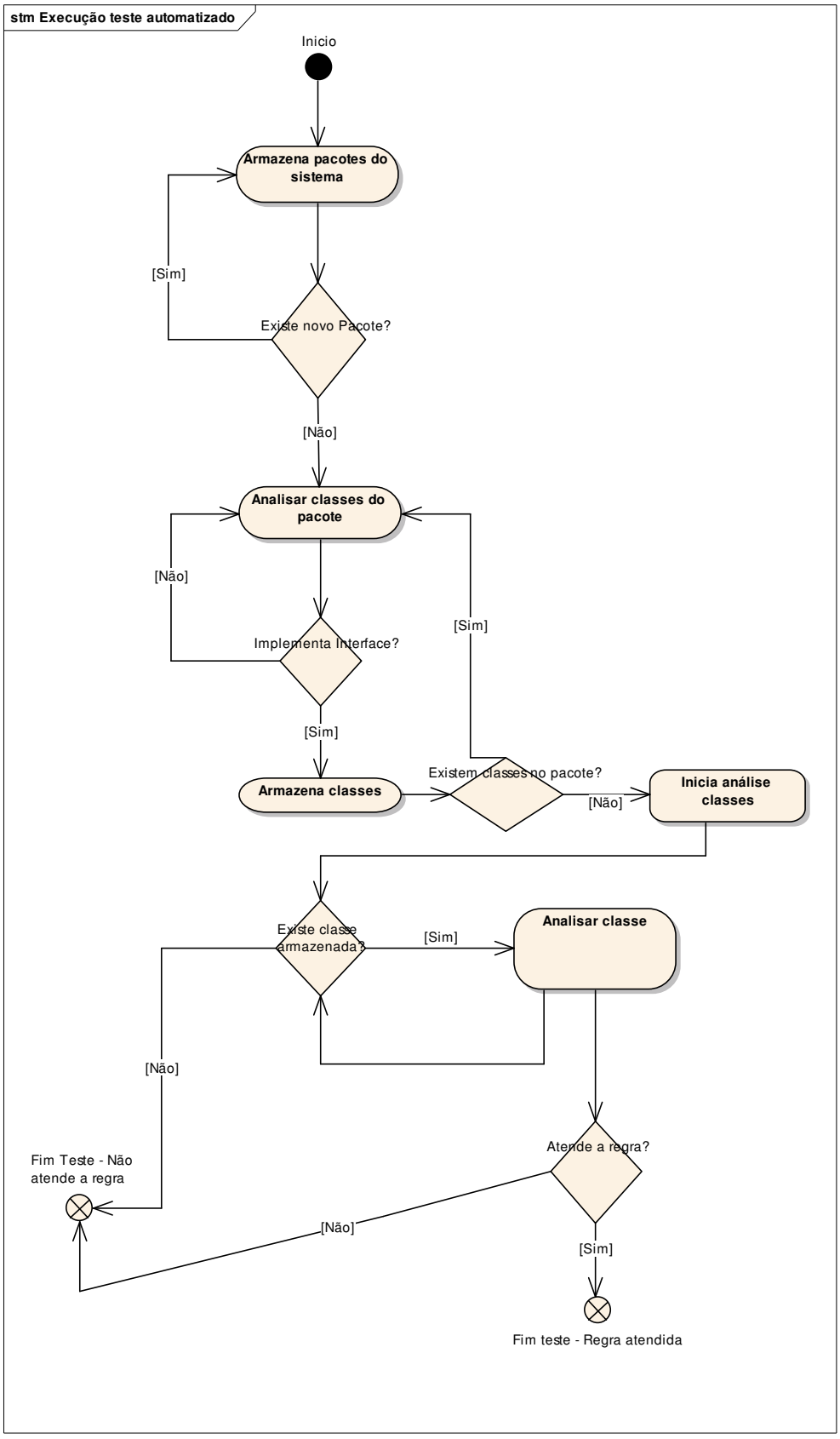


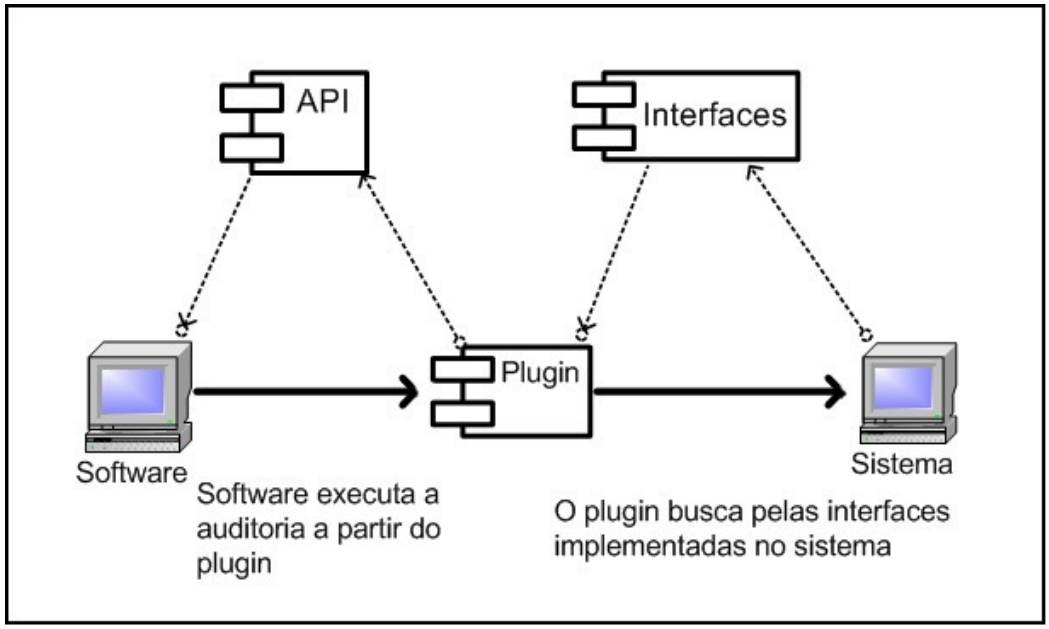
Figura 9 – Diagrama de estados – Execução de teste semi-automatizado

Ao iniciar o teste, são armazenados todos os pacotes informados a partir do arquivo com as configurações iniciais, quando não haver mais pacotes, inicia a verificação das classes.

São verificadas e armazenadas as classes que implementam a interface disponibilizada para verificação da regra. Quando terminar a verificação das classes do pacote, é iniciada a análise das classes, verificando se atendem ou não a regra de segurança.

3.3.2 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 estrutura do SCAN-CC, conforme apresentado na Figura 10.



Fonte: (TRAPP, 2010).

Figura 10 – Estrutura do SCAN-CC

As classes contidas nos pacotes apresentados no decorrer deste capítulo, mostram a estrutura dos testes automatizados com base nos requisitos de auditoria e segurança e identificação e autenticação.

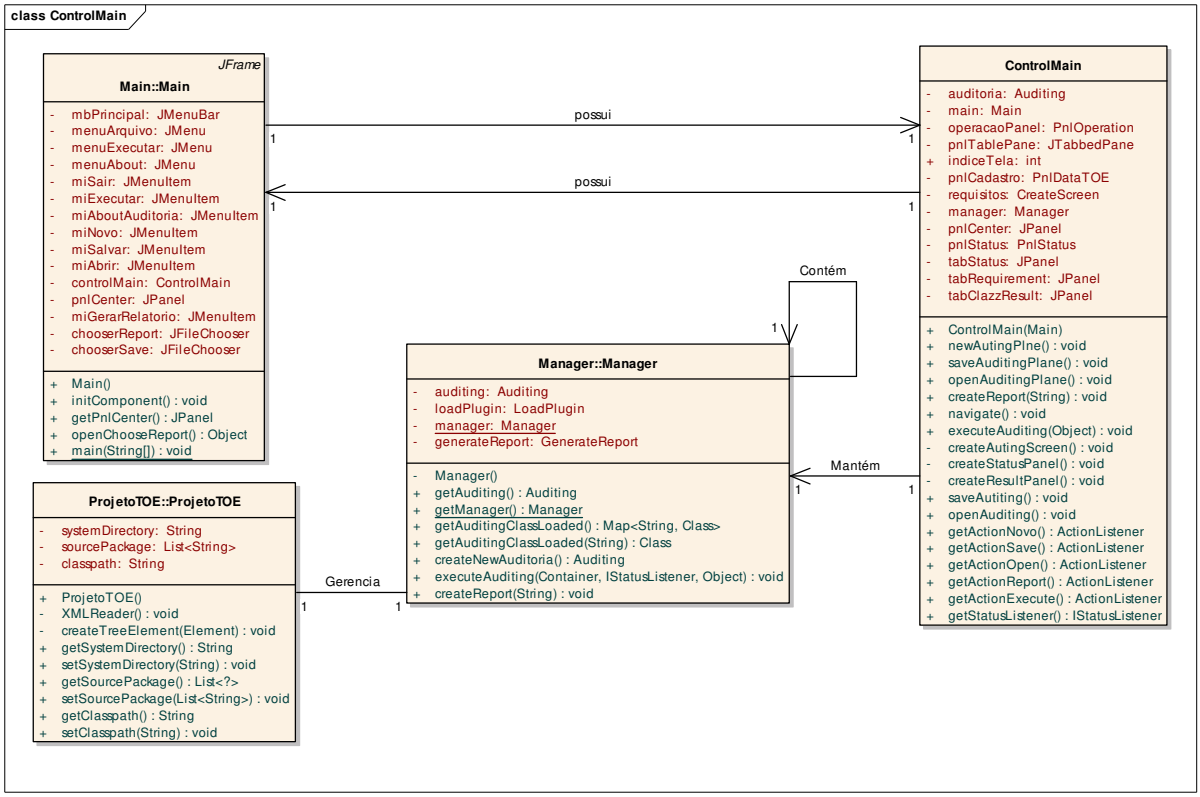


Figura 11– Pacote de execução principal do protótipo

Na Figura 11 são exibidas as classes principais para execução do SCAN-CC. Esta estrutura permanece de acordo com o que foi desenvolvido inicialmente, diferindo apenas a classe `ProjetoTOE`, que foi agregado ao projeto para a realização da leitura de um arquivo em XML com a configuração inicial do projeto que será avaliado. Para um melhor entendimento, segue o detalhamento da classe `ProjetoTOE`:

a) `ProjetoTOE()` – construtor da classe, realiza a chamada do método `XmlReader()`, iniciando o processo de leitura do arquivo;

b) `XmlReader()` – método que realiza a abertura e estrutura as informações do arquivo para que possam ser analisadas;

c) `createTreeElement(Element)` – método que irá buscar as informações do arquivo e armazenar os dados do projeto analisado;

d) `getSysteDirectory()` – retorna o caminho do diretório do sistema;

e) `setSystemDirectory(String)` – seta o diretório do sistema;

f) `getSourcePackage()`– retorna uma lista contendo os pacotes do sistema;

g) `setSourcePackage(String)` – armazena os pacotes informados em uma lista;

h) `getClassPath()`– retorna o endereço do arquivo `.classpath` do sistema;

i) `setClassPath()`– seta o endereço do do arquivo `.classpath` do sistema.

Esta configuração inicial conterá as informações do diretório onde se encontra o projeto, os pacotes contidos neste e o endereço do arquivo `.classpath`. O formato deste arquivo é exibido no Quadro 2.

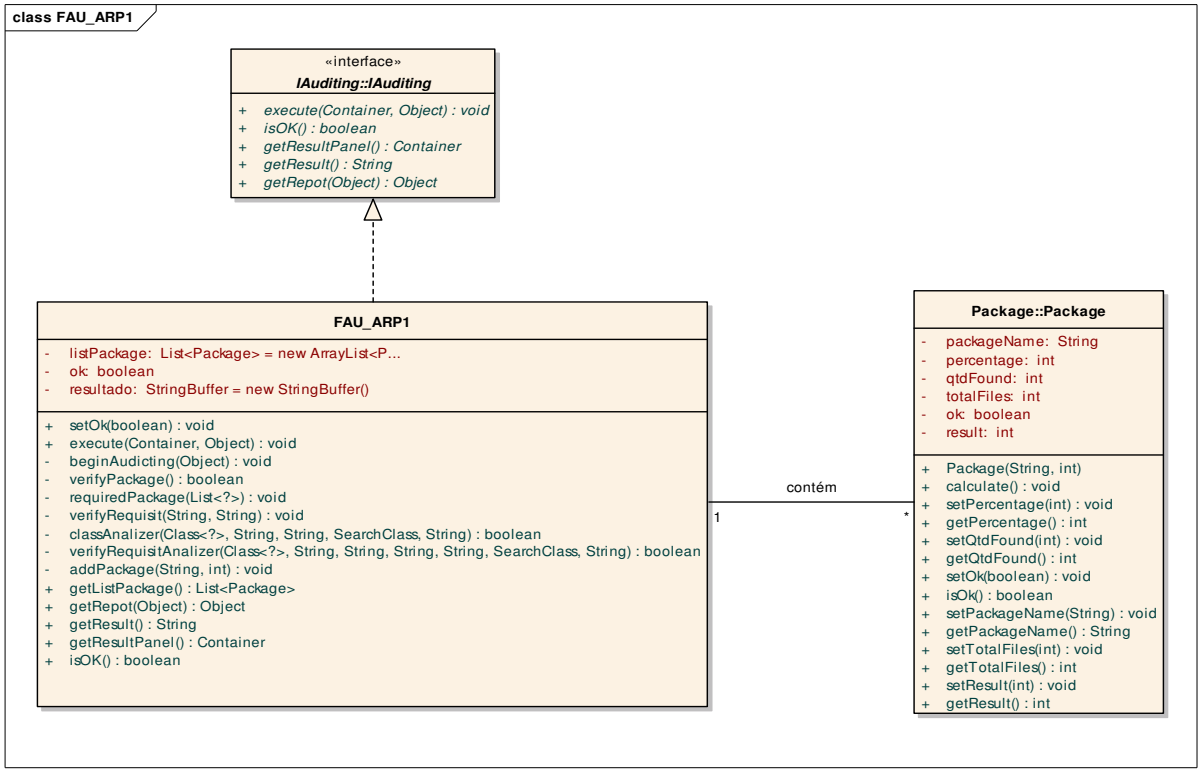


Figura 12– Diagrama de classes requisito de auditoria e segurança FAU_ARP

Como pode ser verificado na Figura 12 e Figura 13, as classes com os testes possuem basicamente a mesma estrutura. Cada uma das classes obrigatoriamente devem implementar a interface `IAuditing`.

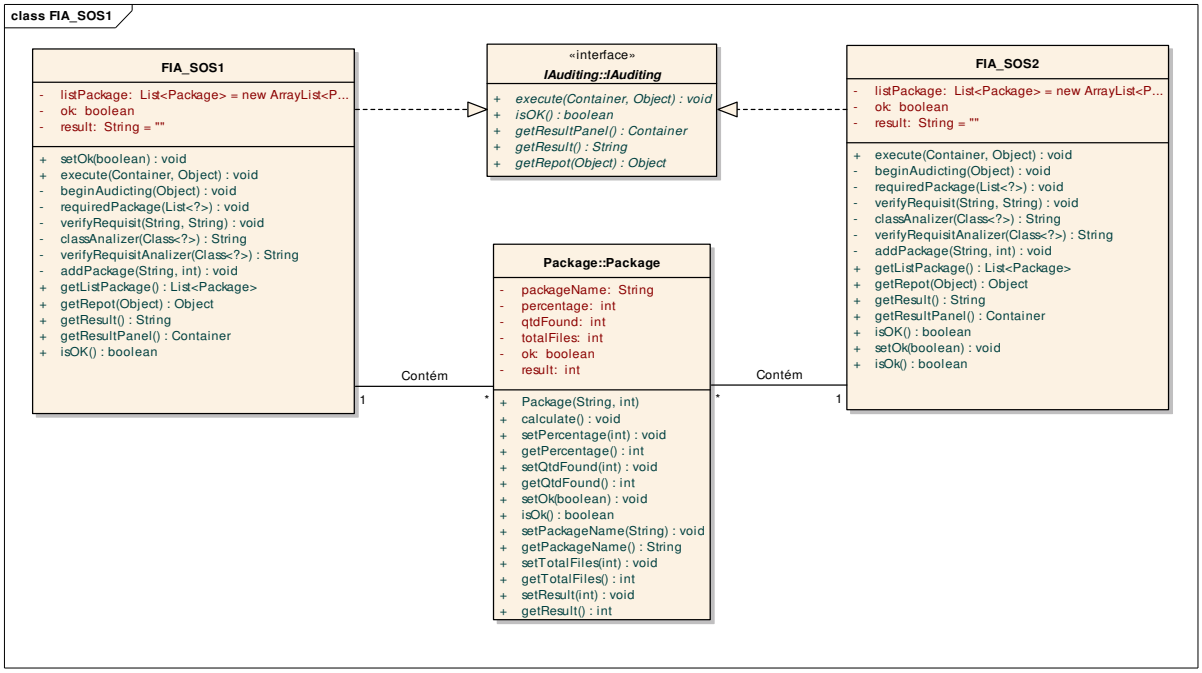


Figura 13 – Diagrama de classes requisito de identificação e autenticação FIA_SOS

Detalhando as classes contidas nos pacotes `FIA_SOS1`, conforme Figura 13:

a) `FIA_SOS1`: esta classe é responsável pela verificação e validação automática do requisito `FIA_SOS1` da norma. Esta classe poderá realizar a verificação em um ou mais pacotes, através do método privado `verifyRequisit`;

b) `FIA_SOS2`: da mesma forma como descrito no item anterior diferindo apenas no requisito a ser verificado, que neste caso é o requisito `FIA_SOS1`;

c) `IAuditing`: interface que deve ser implementada por todos os testes automatizados que foram ou serão implementados. Deve haver uma preocupação no desenvolvimento correto dos métodos implementados, pois todos tem papel fundamental no processo. Os principais métodos são o `execute(Container)`, método que é chamado externamente para inicio da execução da analise do requisito, `isOk()`, método que retorna verdadeiro se o requisito foi verificado com sucesso, onde o TOE atende as regras do requisito e o `getResult`, que retorna o valor do resultado obtido com a analise;

d) `Package`: classe que contém as informações com relação aos pacotes que serão analisados pelo requisito. Existem casos em que o resultado retornado é um valor percentual, verificado também por esta classe.

Na Figura 12, são apresentadas as classes que compõe o requisito `FAU_ARP1` da norma. De forma geral, este pacote possui a mesma estrutura e comportamento citado anteriormente, diferindo apenas a implementação realizada dentro do método

`verifyRequisit`, que é responsável pela execução do teste.

3.4 IMPLEMENTAÇÃO

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

3.4.1 Técnicas e ferramentas utilizadas

Os testes foram desenvolvidos na linguagem Java, seguindo o paradigma da orientação a objetos, utilizando o ambiente de desenvolvimento Eclipse – Galileu e a ferramenta SCAN-CC, agregando os testes implementados.

Para realizar os testes e validar a implementação dos testes, foi desenvolvida uma aplicação que simula o processo de autenticação do usuário, implementando as interfaces definidas.

3.4.2 Dom4j

Para a leitura do arquivo XML com as informações do projeto que será analisado, é utilizada a dom4j, que 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 já utilizado anteriormente para realizar a leitura de um arquivo XML, onde são persistidos todos os requisitos da norma ISO/IEC 15408 que são lidos a cada novo plano de auditoria.

Esta biblioteca também é utilizada para realizar a leitura do arquivo que mantém os dados do projeto que será avaliado.

No Quadro 2 é apresentado parte da estrutura do arquivo XML.

```
<?xml version="1.0" encoding="UTF-8"?>
<Projeto>
<Diretorio>C:\Users\Dionei\Documents\workspace\SistemaTestes\src</Diretorio>
         <Pacotes><Pacote>br.furb.teste</Pacote>     </Pacotes>
         <ClassPath>C:\Users\Dionei\Documents\workspace\SistemaTestes\.classpath
     </ClassPath>
</Projeto>
```

Quadro 2 – Estrutura do arquivo XML do arquivo de entrada do sistema

De forma a manter uma padronização no formato e inserção do caminho do diretório do projeto, pacotes e o *classpath*, deve-se criar um arquivo em XML com a estrutura apresentada anteriormente e este será lido pelo sistema ao iniciar a execução da análise dos requisitos.

As informações deverão ser cadastradas de acordo com cada atributo do modelo apresentado. O arquivo deve possuir uma *tag* principal, chamada de projeto, onde estarão contidas as informações do projeto. No conteúdo da *tag* diretório, dentro do projeto, deverá conter a informação do caminho do diretório do projeto que será analisado. Na *tag* pacotes, serão informados os valores para de cada pacote que deverá ser verificado no projeto e finalizando com o caminho do arquivo *.classpath* do projeto, cadastrado na *tag* classPath.

A leitura do arquivo de informações do sistema a ser auditado é realizada através da classe `ProjetoTOE` e através desta, repassada para todos os requisitos que iram ser verificados.

No Quadro 3 é exibido parte do código que lê o arquivo XML.

```
Private void readElement(Element element) {
for (Iterator<Element> iterator = element.elementIterator(); iterator.hasNext();) {
             Element elementProject = (Element) iterator.next();

             if (elementProject.getName().equalsIgnoreCase("Diretorio")) {
                    this.systemDirectory = elementProject.getStringValue();
                    System.out.println(elementProject.getStringValue());
             } else if (elementProject.getName().equalsIgnoreCase("Pacotes")) {
                    for (Iterator<Element> iterator2 =
elementProject.elementIterator(); iterator2.hasNext();) {
                           Element newElement = (Element) iterator2.next();
                           if (newElement.getName().equalsIgnoreCase("Pacote")) {

       this.sourcePackage.add(newElement.getStringValue());

       System.out.println(elementProject.getStringValue());
                           }
                    }
             } else if (elementProject.getName().equalsIgnoreCase("ClassPath")) {
                    this.classpath = elementProject.getStringValue();
                    System.out.println(elementProject.getStringValue());
             }
       }
}
```

Quadro 3- Parte do código da classe `ProjetoTOE`

3.4.3 Implementação dos testes automatizados.

Na seqüência serão apresentados os testes implementados, que realizam a verificação de conformidades de segurança. No Quadro 4 são apresentados os requisitos das famílias de auditoria de segurança, identificação e autenticação com as suas respectivas verificações.

Requisito	Verificação
FAU_GEN1	Garantir que sejam gerados dados para auditoria
FAU_GEN2	Garantir que o evento de auditoria gerado seja associado ao usuário que gerou o evento
FAU_SEL1	Verificar se o sistema permite a seleção de quais eventos devem gerar dados para auditoria
FAU_SAR1	Verificar se o sistema disponibiliza a leitura da trilha de auditoria para usuários autorizados
FAU_SAA1	Verificar se o sistema aplica um conjunto de regras para indicar uma possível violação das políticas de segurança
FAU_ARP1	Verificar se o sistema executa uma ação depois de detectada uma possível invasão
FIA_AFL1	Verificar se existe uma política de tratamento para sucessivas tentativas de autenticação
FIA_ATD1	Verificar se o sistema mantém uma lista de atributos de segurança por usuário
FIA_SOS1	Verificar se o sistema possui mecanismos para verificação de métricas de senhas
FIA_SOS2	Verificar se o sistema possui um mecanismo que gere senhas dentro de uma métrica

Quadro 4 – Requisitos de auditoria de segurança, identificação e autenticação

3.4.3.1 Teste FAU_GEN1

O requisito FAU_GEN.1 define o nível de eventos de auditoria, e especifica a lista de dados que devem ser registrados para cada registro (COMMON CRITERIA, 2009, p.31). É responsável por garantir que o sistema gere dados de auditoria para cada um dos eventos geradores previamente definidos, como acesso ao sistema e/ou banco de dados, armazenando informações do usuário que gerou o evento e o tipo evento (ALBUQUERQUE E RIBEIRO, 2002, p. 115).

Para a verificação deste requisito, o sistema analisado deverá implementar a interface `IFAU_GENRules`, que contém os métodos que devem ser implementados pelo sistema para que se enquadre aos padrões da norma.

A verificação deste requisito percorre todas as classes que estão dentro do pacote informado pelo auditor no arquivo de configuração e verifica quais classes importam uma API do JAVA próprio para geração de mensagens a partir da aplicação, chamado de `java.util.logging.Logger`, deduzindo que o mesmo armazena dados para auditoria. Além desta verificação, o teste verifica se a classe possui o método `saveDataAudict`, método que tem como objetivo de armazenar as informações da auditoria no banco de dados. No Quadro 5 é exibido parte do código fonte do teste.

```
private void verifyRequisit(String importLogger, String methodSearch,
            String directory) {
boolean classVerify = false;
for (Package package1 : getListPackage()) {
      SearchImport searchImport = searchImport(importLogger,
                  methodSearch, getNameDirectorio(directory, package1));
      try {
      // Calendar date,Object event,String user, String result
      for(int i = 0; i < searchImport.getCLassFound().size();i++){
      classVerify = classAnalizer(Class
.forName(package1.getPackageName()+"."+searchImport.getCLassFound().get(
i).getName().replace(".java","")), methodSearch);
      if(classVerify)
      sb.append("Sucesso FAU_GEN1 - Encontrado em -
"+package1.getPackageName()+"."+searchImport.getCLassFound().get(0).getN
ame());
      else
            sb.append("Falha FAU_GEN1");
      }
} catch (ClassNotFoundException e) {
// TODO Auto-generated catch block
            e.printStackTrace();
      } catch (SecurityException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
      }
      package1.setQtdFound(searchImport.getCLassFound().size());
      package1.setTotalFiles(searchImport.getFileTootal());}}
```

Quadro 5– Parte código fonte do teste automatizado FAU_GEN1

3.4.3.2 Teste FAU_GEN2

De acordo com Albuquerque e Ribeiro (2002, pág. 116), o requisito FAU_GEN2 é responsável por garantir que o evento de auditoria gerado é associado a determinado usuário do sistema.

Para a realização do teste de verificação deste requisito, a classe que deve gerar dados para auditoria deve implementar a interface `IFAU_GENRules`. Para que este requisito seja testado, foi realizada a implementação de um sistema básico de cadastros, onde o usuário que for utilizá-lo deve inserir dados de seu usuário e senha.

Ao iniciar a verificação deste requisito, será solicitada ao auditor qual a quantidade percentual do total de classes do pacote que se espera atender a este requisito. Após ser informado o valor, é iniciado a verificação. Este teste irá realizar a busca em todas as classes contidas no pacote informado no arquivo de configuração inicial, verificando qual destas implementa a interface mencionada anteriormente. Após a busca inicial, será solicitado ao auditor que informe os dados do usuário e senha de um usuário cadastrado no sistema que está

sendo avaliado. Com os dados do usuário, o teste realiza a chamada do método `logginUser` da interface, gerando dados para auditoria com o evento de acesso ao sistema, verificado no requisito anterior. Na seqüência, o teste invoca os métodos `generateEventUser` e `getActualUser` que retornam respectivamente as informações dos dados da auditoria gerados e o usuário atual do sistema que está sendo avaliado. É realizada então a comparação entre o retorno dos dois métodos e se for verificado que o usuário que gerou o evento é o mesmo que o usuário atual do sistema, o atendimento as regras deste requisito são confirmadas. No Quadro 6 é exibido parte do código fonte implementado.

```
private void verifyRequisit(String systemDirectory) {
	// TODO Auto-generated method stub
	boolean classVerify = false;
	for (Package package1 : getListPackage()) {
		SearchClass searchClass = new SearchClass("");
		searchClass.setMethod("logginUser");
		searchClass.search(systemDirectory);
	for (int i = 0; i < searchClass.getCLassFound().size(); i++) {
	try {
	classVerify = classAnalizer(Class.forName(package1
					.getPackageName()   + "."
				+searchClass.getCLassFound().get(i).getName()
				.replace(".java", "")), searchClass.getMethod());
	if (classVerify)
	result.append("Sistema associa usuário com evento de auditoria gerado");
	else
		result.append("Sistema não associa usuário com evento de
auditoria");
	} catch (ClassNotFoundException e) {
		// TODO Auto-generated catch block
		e.printStackTrace();
	}
	}
}
}
```

Quadro 6 - Parte código fonte teste FAU_GEN2

3.4.3.3 Teste FAU_SEL1

Requisito que segundo Albuquerque e Ribeiro (2002, pág. 116), deve verificar se o sistema permite a seleção ou não de quais eventos devem gerar dados para auditoria.

Para a verificação deste requisito, deve ser implementada uma classe *enum* com os tipos de eventos geradores de dados para auditoria. Esta classe deve conter a descrição do evento, um índice para o evento e um atributo do tipo `boolean` informando se este evento deve ou não ser gerado. A classe deve ser desenvolvida conforme Quadro 7.

```
package BR.furb.patterns;

public enum AudictEvents {

	AUTENTICATION(0, "Autenticação", true), //
	IDENTIFICATION(1, "Identificação",true), //
	LOGON(2, "Logon",true),
	LOGOFF(3, "Logoff",true);

	private int indice;
	private String descript;
	private boolean isAudictGenerate;

	private AudictEvents(int ind, String value, boolean isAudict){
		this.indice = ind;
		this.descript = value;
		this.isAudictGenerate = isAudict;
	}

	public static String getEventAudict(int ind){
		switch (ind) {
			case 0:     return new String ("Autenticação");
			case 1:     return new String ("Identificação");
			case 2:     return new String ("Logon");
			case 3: return new String ("Logoff");
		default: return null;
		}
	}
	public void setIsEventAudictGenerator(boolean val){
		this.isAudictGenerate = val;
	}
	public boolean isEventAudictGenerator(){
		return this.isAudictGenerate;
	}

}
```

Quadro 7- Classe do tipo enum para configurar os tipos de eventos

O teste realiza a busca das classes, verificando qual delas implementa a interface `IFAU_SELRules` com regras para este requisito. Ao encontrá-las o teste verifica quais são os eventos que estão cadastrados através do método `getAudictEvents`. Após realizar esta verificação, é realizada a alteração do evento verificado e realizada novamente a consulta. No Quadro 8, é exibido de forma mais detalhada o processo de verificação.

```
Private boolean classAnalizer(Class<?> clazz, String method) {
            boolean ret = false;
            try {
                  Method methodExec = clazz.getMethod(method,new Class[] {
int.class });
                  Object c = clazz.newInstance();
                  //verifica o valor inicial para o evento, se deve ou nao
gerar dados para auditoria
                  Object invoke = methodExec.invoke(c, 1);
                  //verifica se exite o método para alteração do valor do
evento, desativar a geração de dados para auditoria
                  Method methodExecSet =
clazz.getMethod("setValueEventGenerate",
                                                new Class[]{int.class,
boolean.class });
                  //desativa geração de dados para auditoria
                  Object newInvoke = methodExecSet.invoke(c, 1, false);
                  //verifica se foi armazenada alteração
                  Object invokeNew = methodExec.invoke(c, 1);

                  if (invoke != invokeNew){
                        ret = true;
                  }else
                        ret = false;

            } catch (SecurityException e) {
                  // TODO Auto-generated catch block
                  e.printStackTrace();
            } catch (NoSuchMethodException e) {
                  // TODO Auto-generated catch block
                  e.printStackTrace();
            } catch (InstantiationException e) {
                  // TODO Auto-generated catch block
                  e.printStackTrace();
            } catch (IllegalAccessException e) {
                  // TODO Auto-generated catch block
                  e.printStackTrace();
            } catch (IllegalArgumentException e) {
                  // TODO Auto-generated catch block
                  e.printStackTrace();
            } catch (InvocationTargetException e) {
                  // TODO Auto-generated catch block
                  e.printStackTrace();
            }

            return ret;
      }
```

Quadro 8 - Parte do código de analise do requisito FAU_SEL1

Caso a alteração tenha ocorrido de acordo com o processo, o teste retorna uma mensagem de confirmação.

3.4.3.4 Teste FAU_SAR1

As funções de segurança do TOE deve prover a usuários autorizados a capacidade de

ler os dados de auditoria gerados (ALBUQUERQUE; RIBEIRO, 2002, p. 117).

Para que este requisito seja testado, a classe responsável pela exibição dos dados da auditoria deve implementar a interface `IFAU_SARRules` com o modelo de desenvolvimento que deve ser seguido.

O teste realiza a busca por classes que contenham a interface citada anteriormente e solicita os dados de acesso ao sistema do usuário que deve ter a capacidade de leitura da trilha de auditoria. Após a inserção dos dados solicitados, é verificado se nas classes encontradas existe um método para autenticação no sistema. Ao encontrar, é realizado o processo de autenticação no sistema no sistema analisado. Com o método `isAdministrator` é realizada a verificação do tipo do usuário se é administrador ou não. Após esta verificação, são solicitadas as informações da trilha de auditoria através do método `getAuditData` e retornando um valor válido o teste é validado. Este processo é apresentado de forma mais detalhada na parte do código fonte apresentado na Quadro 9.

```
Private String verifyRequisitAnalizer(Class<?> forName, String userAdmin,
                  String pass) {
Method methodExec, newMethod = null;
String string = null;   String metodo = null;   Object c = null;
Method methods[] = forName.getMethods();
for (int i = 0; i < methods.length; i++) {
      if (methods[i].getName().contains("loggin")) {
            metodo = methods[i].getName();
                  break;      }     }
      try {
c = forName.newInstance();
newMethod = forName.getMethod(metodo, new Class[] { String.class,
                  char[].class });
Object retorno1 = newMethod
            .invoke(c, userAdmin, pass.toCharArray());
methodExec = forName.getMethod("isAdministrator", new Class[] {});
Object retorno = methodExec.invoke(c, null);
      newMethod = null;
newMethod = forName.getMethod("getAuditData", new Class[] {});
Object ret = newMethod.invoke(c, null);
if (Boolean.valueOf(retorno.toString())
            && !ret.toString().equalsIgnoreCase(" ")) {
string = "Requisito atendido. Permite a leitura dos dados da auditoria
pelo usuário      auditor";
this.ok = true;
} else {string = "Não permite a leitura dos dados da auditoria pelo
usuário administrador";
this.ok = false;              }
} catch (SecurityException e) {
e.printStackTrace();
return string;)
```

Quadro 9 – Trecho do código fonte que verifica o requisito FAU_SAR1

3.4.3.5 Teste FAU_SAA1

Conforme Albuquerque e Ribeiro (2002, p. 121) o requisito FAU_SAA1, as funções de segurança devem ser capazes de aplicar um conjunto de regras na monitoração dos eventos auditados e com base nestas regras indicar violação da política de segurança do sistema.

Para que o teste possa ser realizado, o sistema deve implementar as interfaces `IFAU_SAARules` e IFAU_GENRules.

O teste verifica e armazena todas as classes do pacote informado que implementam as interfaces citadas. Após esta verificação, realiza a busca pelo método `logginUser` e realiza sua execução. A cada execução deste método, o teste informa ao sistema verificado dados de autenticação inválidos, forçando a geração de dados de auditoria por erro no acesso. A cada execução o teste verifica o método `isInvasion` e após uma série de tentativas de acesso incorretas e o retorno do método mencionado for verdadeiro, o teste foi realizado com sucesso. No Quadro 8 é apresentado parte do código do teste que realiza esta verificação.

```
Private boolean verifyRequisitAnalizer(Class<?> forName, String user,
            String pass, String methodSearch) {
Method methodExec, newMethod = null;
boolean ret = false;
try {
      methodExec = forName.getMethod(methodSearch, new Class[] {
                              String.class, char[].class });
      Object c = forName.newInstance();
      for (int i = 0; i < 5; i++) {
            methodExec.invoke(c, user, pass.toCharArray());
            newMethod = forName.getMethod("isInvasion", new Class[] {});
            Object inv = newMethod.invoke(c, null);
            if (inv.toString().equals("true"))
                  ret = true;
            else
                  ret = false;
            }
            setOk(ret);

            } catch (SecurityException e) {
                  // TODO Auto-generated catch block
                  e.printStackTrace();
            } catch (IllegalArgumentException e) {
                  // TODO Auto-generated catch block
                  e.printStackTrace();
            } catch (IllegalAccessException e) {
                  // TODO Auto-generated catch block
```

Quadro 8 – Parte código fonte que realiza a verificação do requisito FAU_SAA1

3.4.3.6 Teste FAU_ARP1

O requisito FAU_ARP1, conforme Albuquerque e Ribeiro (2002, p. 124), define que as funções de segurança devem executar uma lista de ações na detecção de uma possível violação.

Para que o teste ser realizado, o sistema deve implementar as interfaces `IFAU_ARPRules` e `IFAU_GENRules`.

De acordo com padrão definido na interface, as funções de segurança devem enviar um email para o administrador do sistema assim que for detectada a tentativa de invasão. Para verificar se as funções de segurança do TOE atendem a este requisito, o teste força a tentativa de invasão, realizando a múltipla tentativa de acesso ao sistema sem sucesso. O teste verifica e armazena todas as classes do pacote informado que implementam as interfaces citadas. Após esta verificação, realiza a busca pelo método `logginUser` e realiza sua execução. A cada execução deste método, o teste informa ao sistema verificado dados de autenticação inválidos, forçando o erro de tentativa de invasão após cinco tentativas de acessar ao sistema e sem sucesso. Após ter forçado a geração do erro por invasão, o teste verifica o método `executeSendEMailToAdmin` que deve retornar verdadeiro quando foi enviado o email para o administrador do sistema. No Quadro 9 é apresentado parte do código do teste que realiza esta verificação.

```
For (int k = 0; k < searchInterface.getCClassFound().size(); k++) {
    Class<?> interfaces[] = forName.getInterfaces();
for (Class class1 : interfaces) {
if (class1.getSimpleName()
      .equalsIgnoreCase(interfaceSearch)) {
implementsInterface = true;
Method methods[] = class1.getMethods();
for (Method method : methods) {
      newMethod = forName.getMethod(method.getName(),
      new Class[] {});
Object c1 = forName.newInstance();
Object inv = newMethod.invoke(c1, null);
if (inv.toString().equals("true")) {
    ret = true;
} else
      ret = false;
      setOk(ret);
}}}
```

Quadro 10 – Parte código fonte que realiza a verificação do requisito FAU_ARP1

3.4.3.7 Teste FIA_AFL1

Conforme Albuquerque e Ribeiro (2002, p. 139) o requisito FIA_AFL1, define que deve haver uma política de tratamento para tentativas falhas de autenticação sucessivas.

Para que o teste ser realizado, o sistema deve implementar as interfaces `IFIA_AFLRules`.

De acordo com o que é definido nesta interface, as funções de segurança devem implementar um método que retorna se determinado usuário esta ou não bloqueado através do método `isUserBloqued`. O teste irá percorrer todas as classes existentes no pacote informado, verificando se a classe implementa a interface citada. Após esta verificação, realiza a simulação do processo de múltiplas tentativas de acesso. Para este requisito, foi definido que a quantidade máxima de tentativas de acesso consecutivas seria cinco. Após a quinta tentativa incorreta, ao verificar o método mencionado acima, o valor deve ser verdadeiro. O método `getProcedToDesbloq` que deve ser implementado pelas funções de segurança do TOE, para o que seja informado ao usuário que foi bloqueado quais os procedimentos que ele deve tomar para desbloquear sua conta.

3.4.3.8 Teste FIA_ATD1

Conforme Albuquerque e Ribeiro (2002, p. 141) o requisito FIA_ATD1, define que as funções de segurança devem manter uma lista de atributos de segurança para cada usuário.

Para que o teste ser realizado, o sistema deve implementar as interfaces `IFIA_ATDRules`.

Este teste realiza a verificação das classes que implementam esta interface e verifica se a classe implementa métodos que retornam o valor dos atributos de segurança que devem ser armazenados por usuário. O método `getUserAtentication e getPassword` verificam os dados de acesso do usuário. O teste irá verificar se há um tratamento nas funções de segurança que implementam esta interface para garantir a atualização dos dados para autenticação do usuário. Através do método `getDiasValidadeAutentic` será verificado qual a quantidade em dias de intervalo que seja realizada a atualização dos dados. Sendo implementados estes métodos, é garantida a quantidade mínima de requisitos para o atendimento desta regra.

3.4.3.9 Teste FIA_SOS1

Conforme Albuquerque e Ribeiro (2002, p. 141) o requisito FIA_SOS1, define que as funções de segurança devem possuir mecanismos para garantir que as senhas atendam a determinada métrica.

Para que o teste ser realizado, o sistema deve implementar as interfaces `IFIA_SOSRules`.

Para a verificação deste requisito, são verificadas quais classes implementam a interface citada anteriormente. Assim que for realizado este processo, é solicitado ao auditor que informe usuário e senha do sistema a ser avaliado. Após a informação destes dados, este teste irá verificar a compatibilidade dos dados informados através do método `validaPass`, que devem ser desenvolvidos de forma a verificarem se a senha informada atende a requisitos como tamanho mínimo e inclusão de caracteres numéricos e alfanuméricos. Este teste verifica o retorno deste método, que se for verdadeiro, assume que a classe atende a este requisito.

3.4.3.10 Teste FIA_SOS2

Conforme Albuquerque e Ribeiro (2002, p. 143) com relação ao requisito FIA_SOS2, define que as funções de segurança devem fornecer um mecanismo que gere senhas dentro de uma métrica pré definida.

Para que o teste ser realizado, o sistema deve implementar as interfaces `IFIA_SOSRules`.

Este teste inicia, como os anteriores, verificando as classes que implementam esta interface e realiza a busca pelo método `generatePassword` que deve ser responsável por gerar senhas dentro da métrica. Esta métrica deve estar corretamente implementada no método `validaPass`. O teste solicita uma senha através do método `generatePassword` e realiza a sua validação através do método `validaPass`. Se o retorno deste método for verdadeiro, assume-se que a classe que esta sendo verificada atende ao requisito testado.

3.4.4 Operacionalidade da implementação

Nesta seção será apresentada a operacionalidade do software desenvolvido. Ao iniciar a ferramenta SCAN-CC, esta carrega os todos *plugins* implementados. Para que sejam realizados os testes, o auditor inicia o processo de criação de um novo plano de auditoria, inserindo os dados necessários, selecionando os requisitos que serão verificados, conforme Figura 164, 15 e 16.

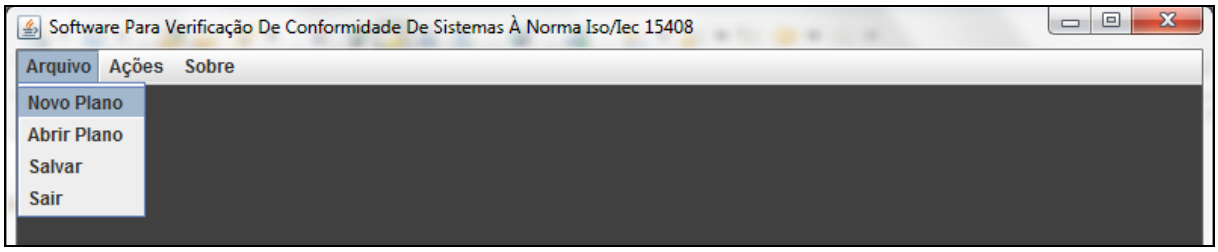


Figura 14 – Criando um novo plano de auditoria

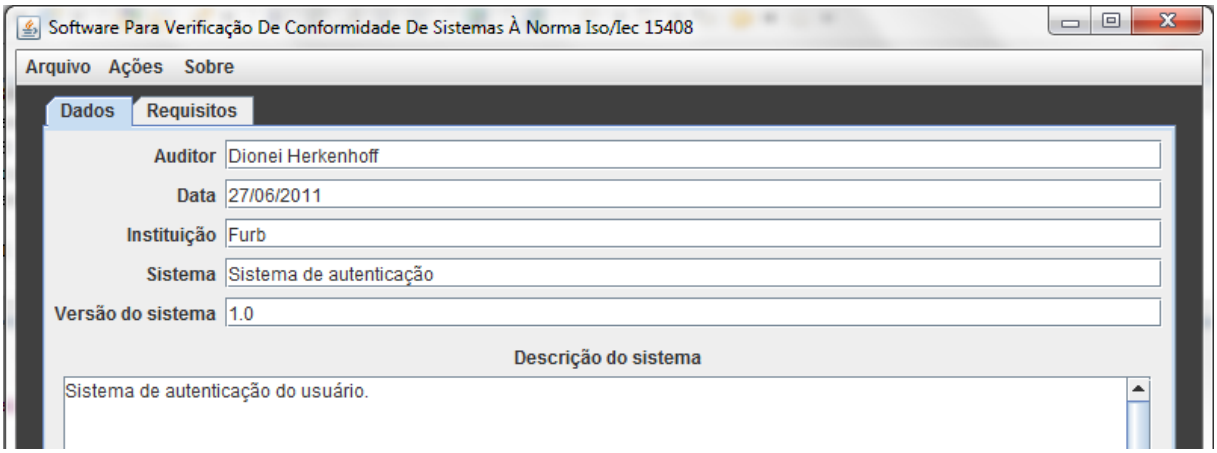


Figura 15 – Janela para digitação do dados da auditoria

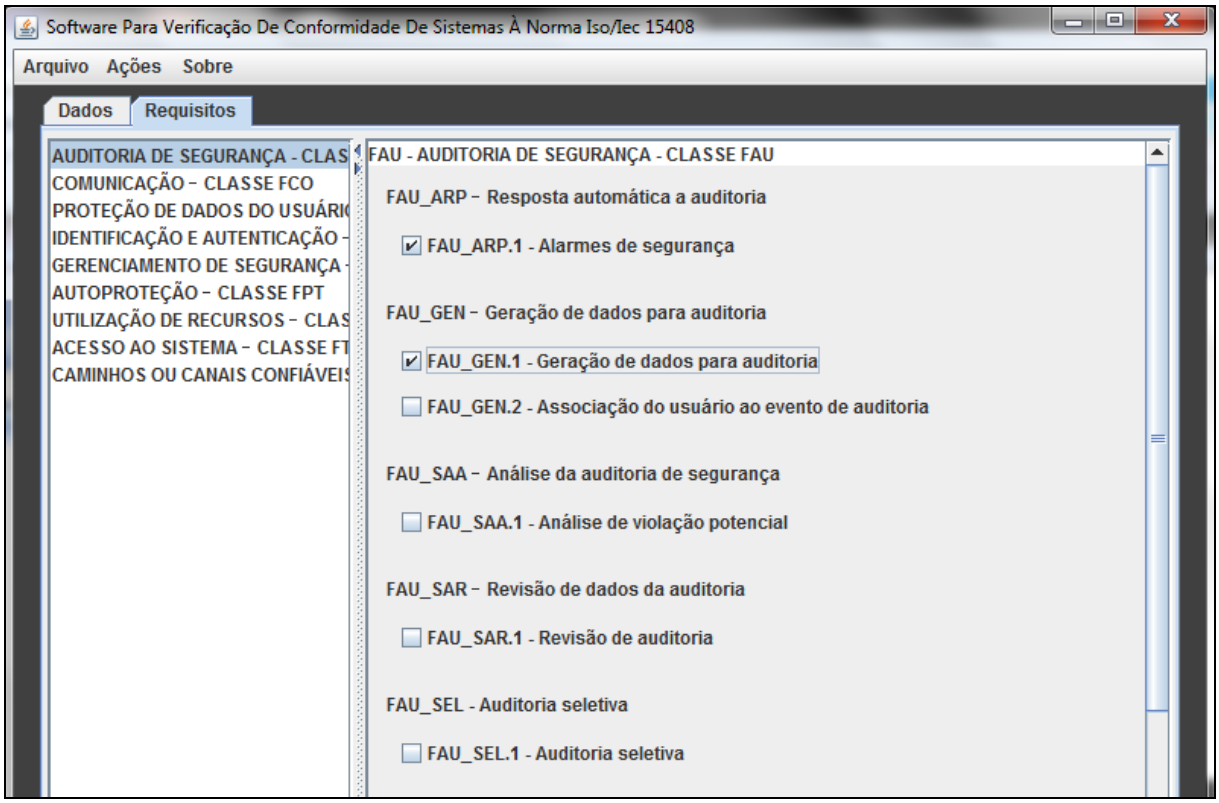


Figura 16 – Selecionando requisitos que serão verificados

Após o término destes passos, o auditor inicia a execução dos testes. Inicialmente o SCAN-CC irá solicitar o arquivo XML com as configurações do sistema que está sendo auditado, conforme Figura 17.

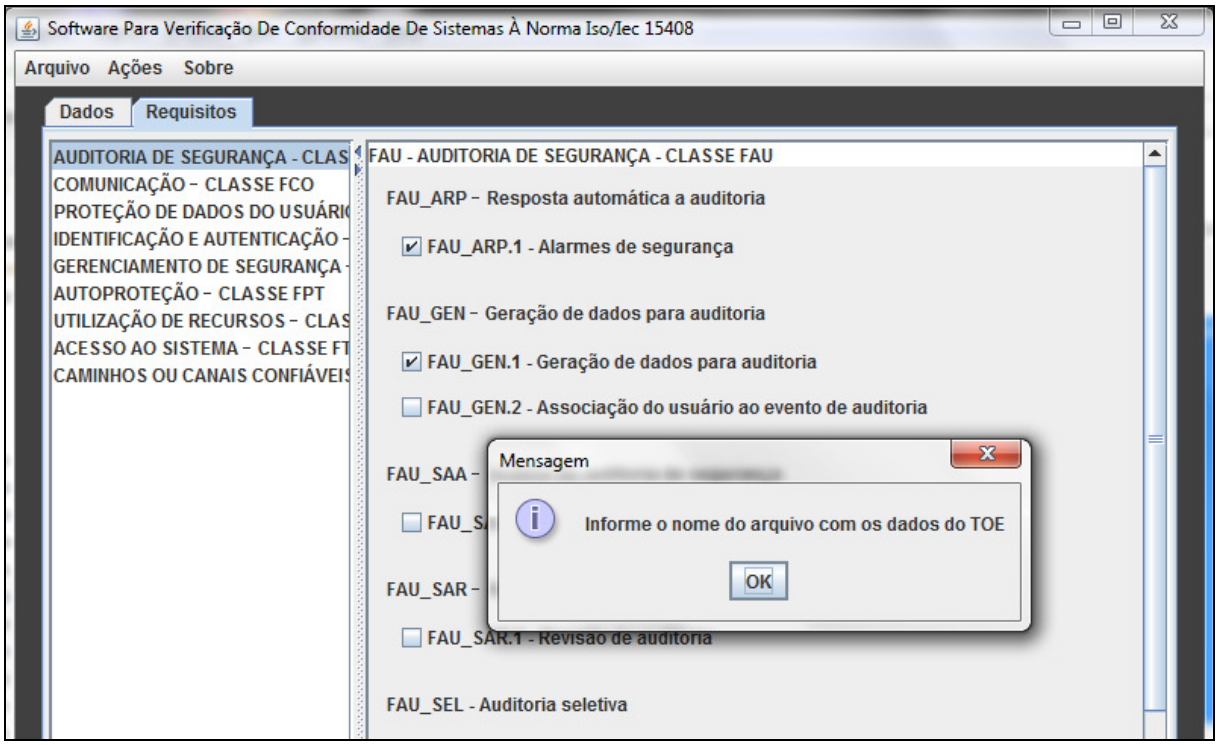


Figura 17 – A ferramenta SCAN-CC solicitando o arquivo XML com os dados do sistema auditado

No teste FAU_GEN1 e FAU_GEN2, ao ser executado, é solicitado ao auditor que informe a quantidade em percentual de quanto das classes do TOE em que devem ser encontradas as informações para geração de auditoria, Figura 18.

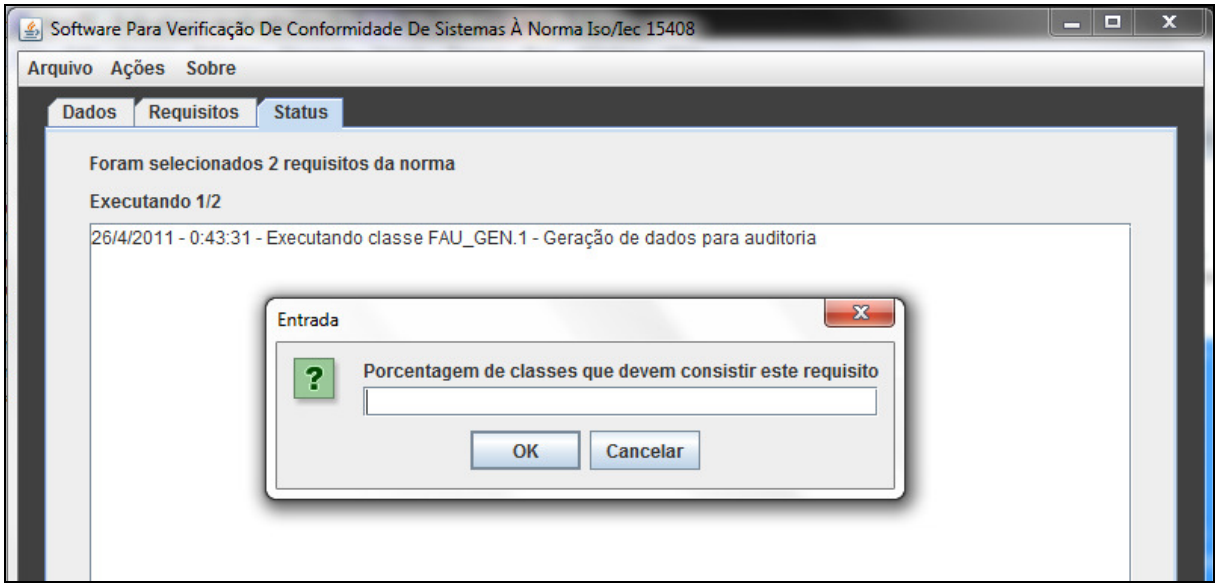


Figura 18 – Informação solicitada pelos testes FAU_GEN1 e FAU_GEN2

Após ser informado pelo auditor o percentual desejado, o teste é executado e o resultado é exibido.

No teste FAU_ARP1, ao ser executado, o teste força a tentativa de invasão por tentativas múltiplas de autenticação e realiza a verifica se o TOE define alguma ação a ser tomada caso for identificada uma potencial invasão, como por exemplo enviar um email para o administrador do sistema informando a tentativa de invasão. Na Figura 19 é exibido o resultado gerado do teste.

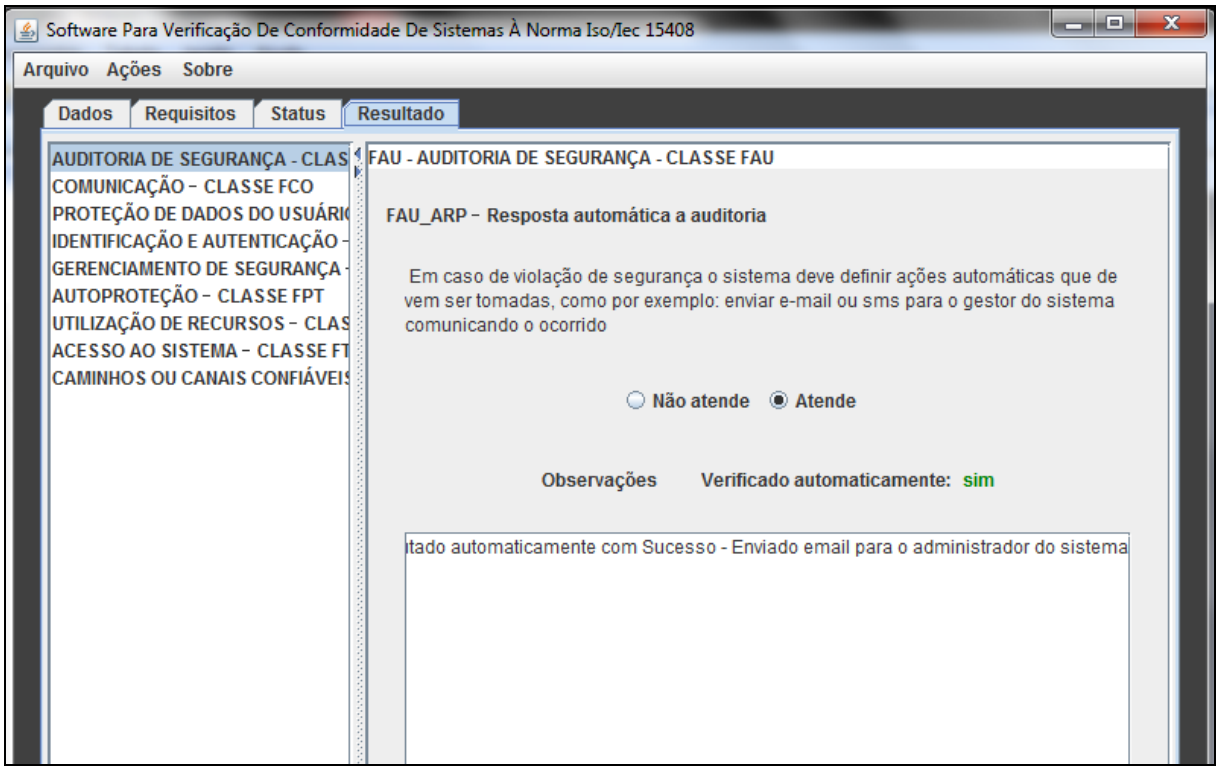


Figura 19 – Resultado do teste FAU_ARP após ser executado

No teste FAU_SAR1, é solicitado ao auditor que informe o usuário e senha do administrador ou o usuário com permissão para ler a trilha de auditoria.

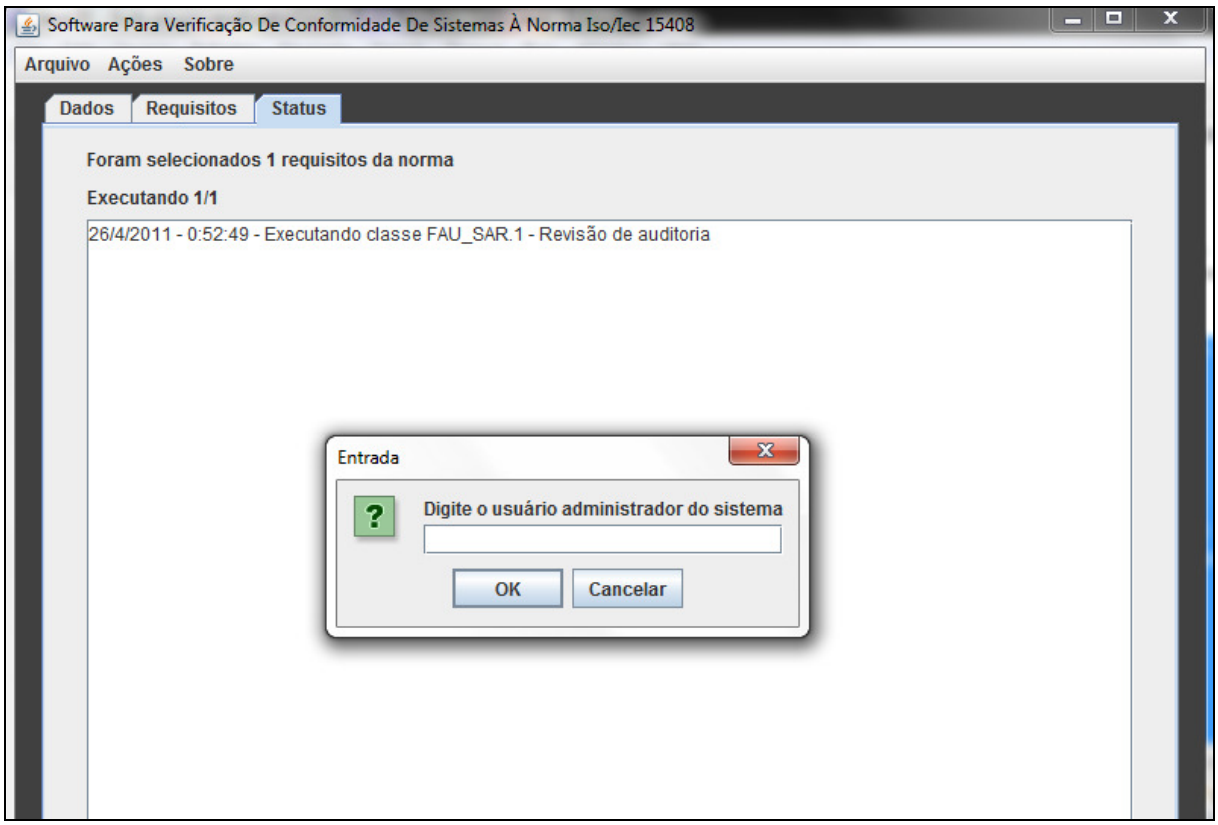


Figura 20 – Solicita ao auditor os dados de usuário e senha do usuário com permissão de leitura da trilha

No teste FAU_SAA o teste verifica se o TOE possui mecanismos que reconheçam uma tentativa de invasão. O teste realiza tentativas consecutivas de autenticação falha e verifica se o há retorno positivo do TOE para tentativa de invasão. O teste irá verificar se houve tentativa de invasão através do método `isInvasion`. O retorno sendo positivo exibe a mensagem confirmando a conformidade do TOE, conforme Figura 21.

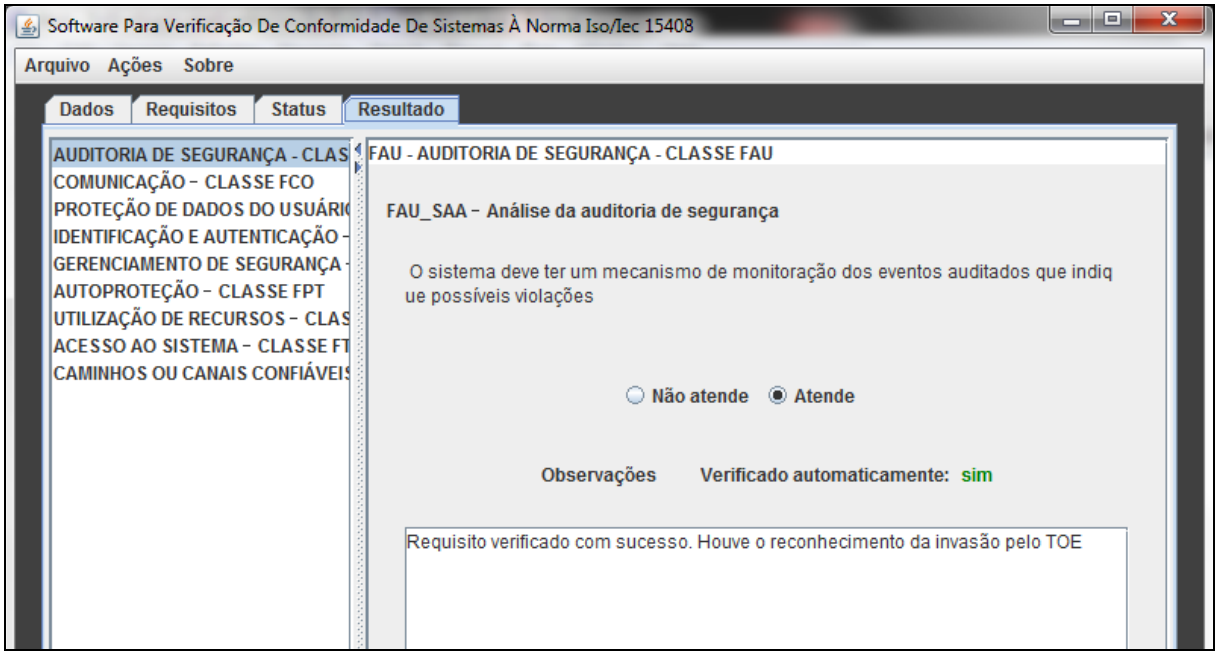


Figura 21 – Resultado teste FAU_SAA1

No teste FAU_SEL1 verifica se o TOE permite a seleção de eventos que irão gerar dados para auditoria. O teste verifica quais são os eventos que geram dados para auditoria através do método `getAudictEvents`. Após esta verificação, seleciona um determinado evento através do método `setValueEventGenerate`. Realiza novamente a verificação dos eventos geradores de dados para auditoria verificando se o evento selecionado anteriormente está dentro dos que foram verificados. Caso esteja exibe mensagem positiva conforme Figura 22.

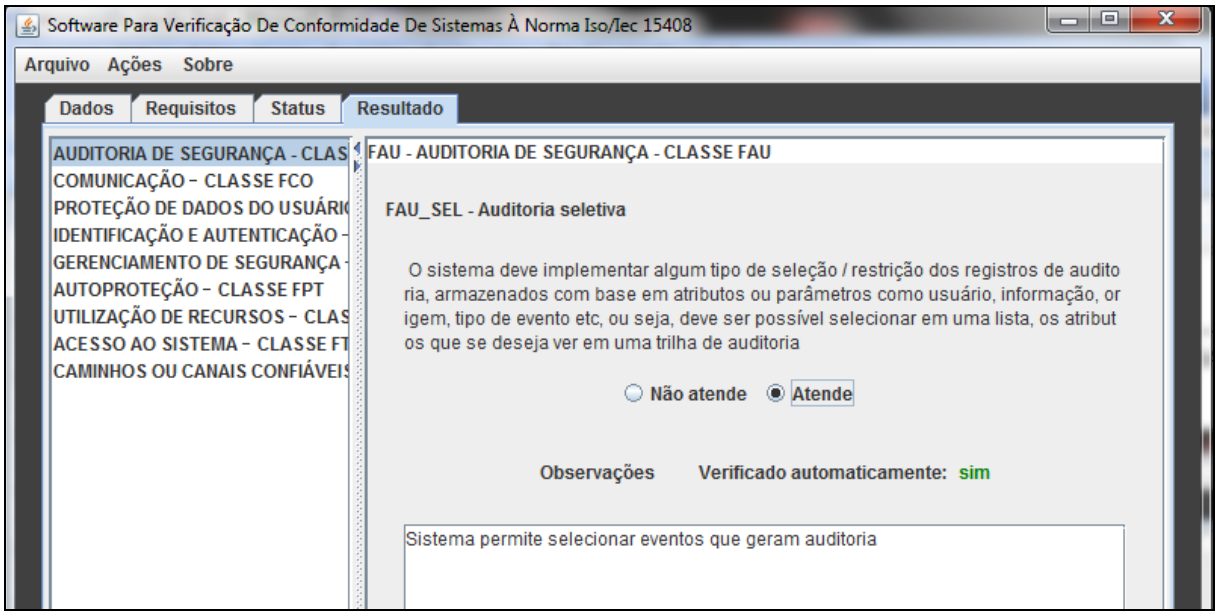


Figura 22 – Resultado teste FAU_SEL1

No teste FIA_AFL, o teste solicita ao auditor que informe um usuário e senha inválidos, Figura 23 e Figura 24.

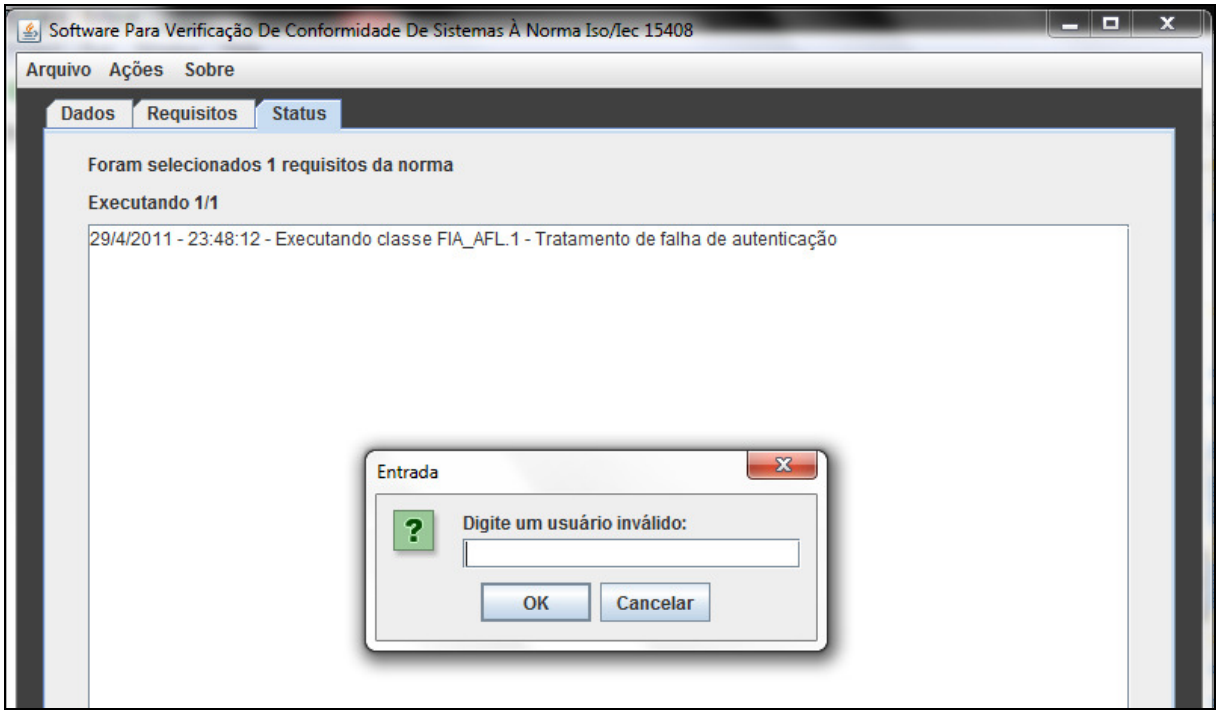


Figura 23 – Solicita usuário inválido

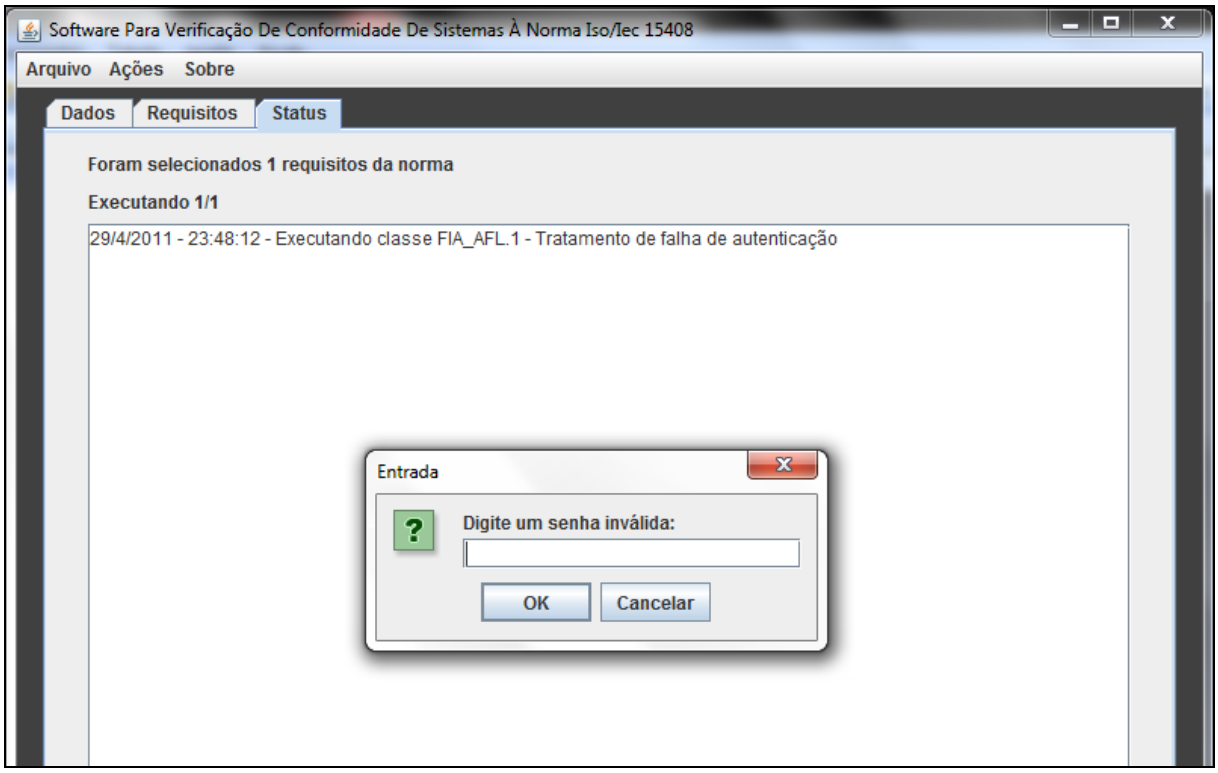


Figura 24 – Solicita senha invalida

Após o auditor confirmar, o teste inicia a execução forçando a geração de um erro por quantidade de tentativas falhas de acesso ao sistema com os dados informados pelo auditor.

No teste FIA_ATD, o teste verifica se o TOE implementa os métodos necessários para o armazenamento das informações do usuário do sistema. Depois de realizada a verificação, caso positivo, retorna a mensagem exibida na Figura 25.

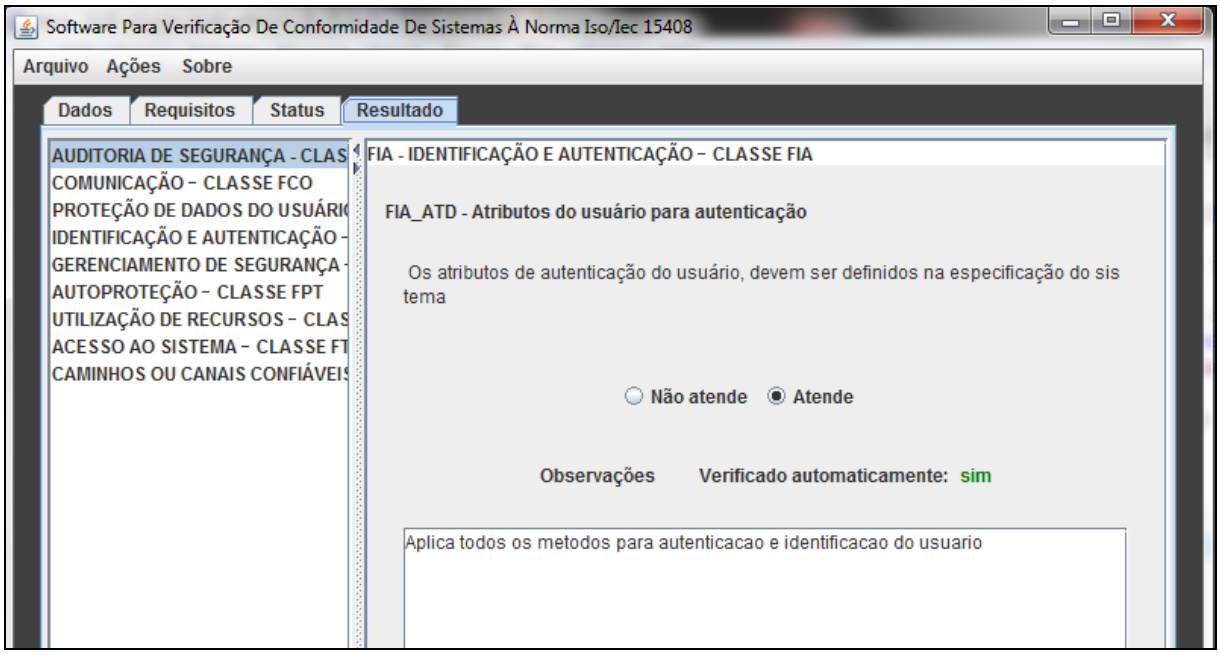


Figura 25 – Teste FIA_ATD

O último teste em que se faz necessária a interação com o auditor é o teste FIA_SOS1,

onde são solicitados informações de usuário e senha (Figura 26 e 22). Com essas informações este teste irá verificar se os dados informados atendem à métrica de senhas do sistema auditado.

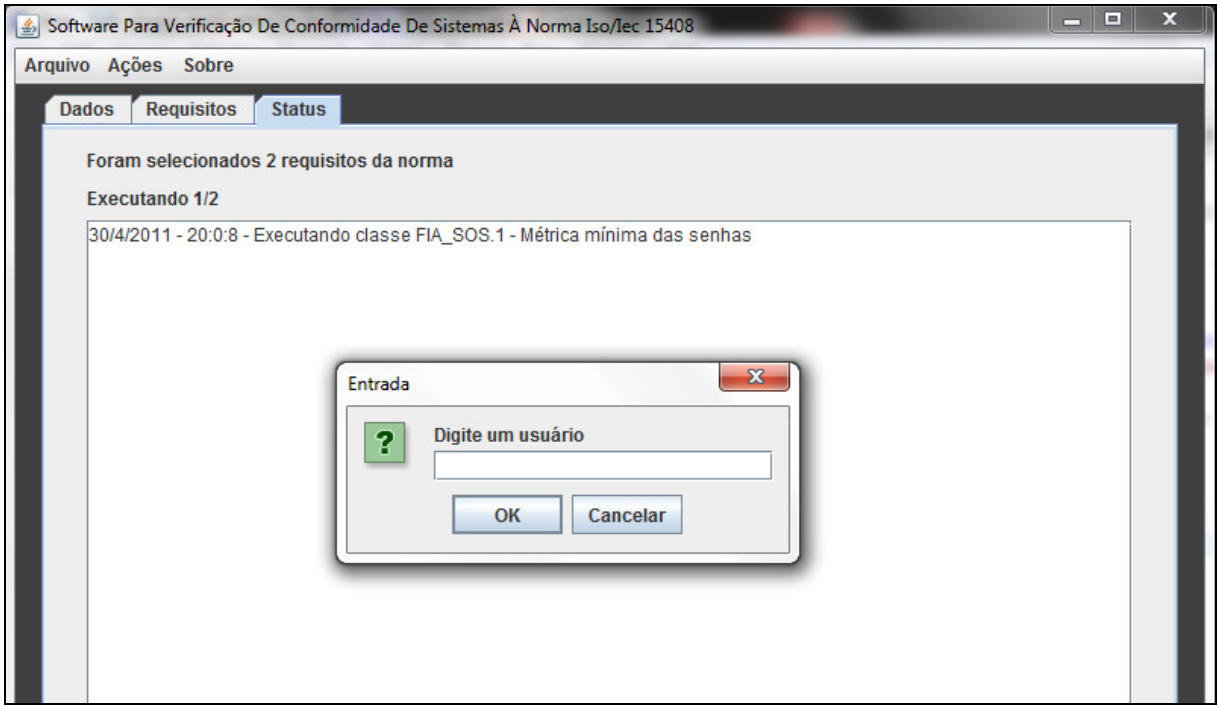


Figura 26 – Solicitação do usuário pelo teste FIA_SOS

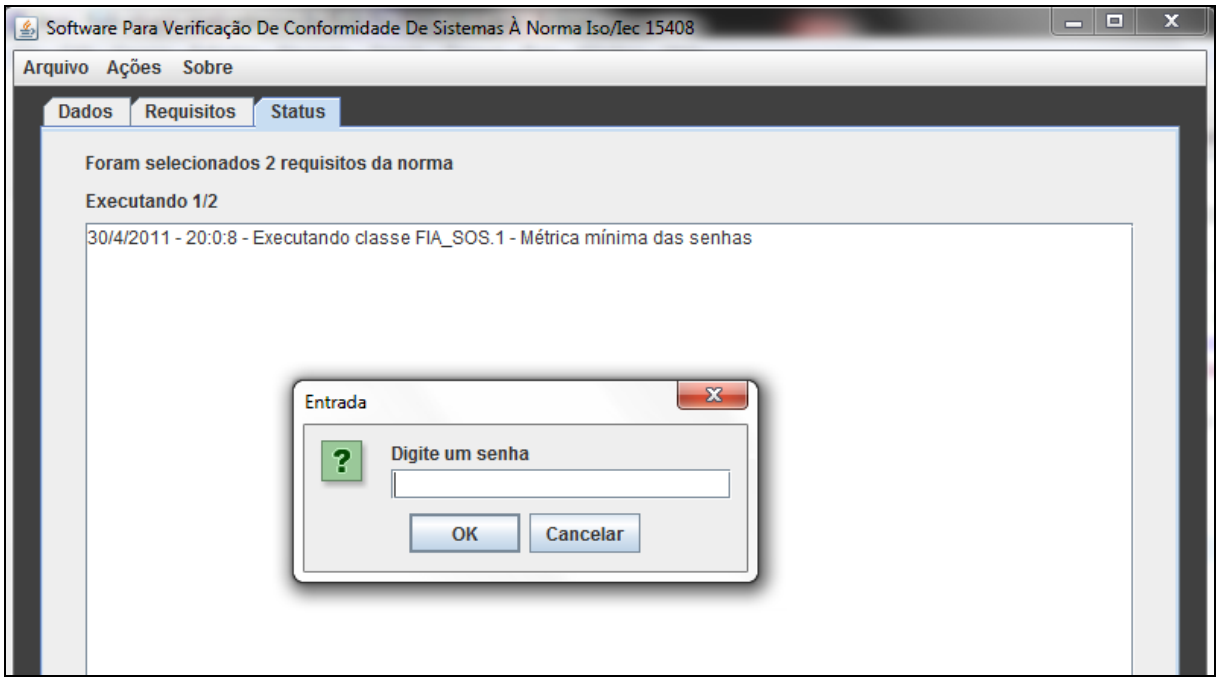


Figura 27 – Solicitação de senha pelo teste FIA_SOS

Na Figura 28, é apresentado parte do relatório gerado com o resultado dos testes realizados.

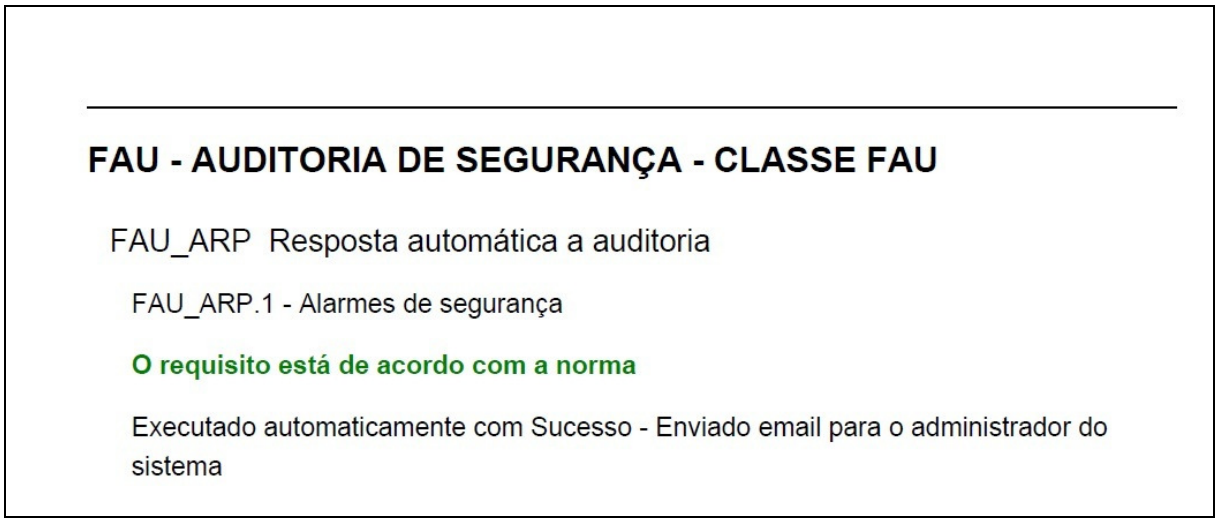


Figura 28 – Parte do relatório com resultado dos testes.

3.5 RESULTADOS E DISCUSSÃO

Com o término deste trabalho pode se realizar verificações automatizadas de requisitos de auditoria de segurança e identificação e autenticação com base na norma ISO/IEC 15408. Entretanto, houve uma grande dificuldade para encontrar uma forma de realizar os testes de uma forma mais genérica. Visando contornar este problema, foram desenvolvidas interfaces que são disponibilizadas para que o sistema a ser verificado, forçando assim o desenvolvimento dentro dos padrões descritos nos requisitos de segurança da norma. Uma das vantagens de se trabalhar desta forma é que não há a necessidade de realizar testes utilizando análise estática, mas sim utilizando reflexão computacional, que possibilita além da introspecção, também realizar a analise e invocar métodos de outras classes em tempo de execução. No Quadro 10, é exibido um comparativo, mostrando quais testes foram agregados à ferramenta com a implementação realizada.

Requisito	SCAN-CC anterior			SCAN-CC atual		
	Atende	Atende parcialmente	Não Atende	Atende	Atende parcialmente	Não Atende
FAU_ARP			X	X		
FAU_GEN		X		X		
FAU_SAA			X	X		
FAU_SAR			X	X		
FAU_SEL			X	X		
FIA_AFL			X	X		
FIA_SOS			X	X		
FIA_ATD				X		
FIA_UAU	X			X		

Quadro 10 – Comparativo SCAN-CC

Com relação aos trabalhos correlatos, *IBM Rational® AppScan® Express Edition,* ferramenta comercial da IBM que realiza verificações de segurança em softwares desenvolvidos para a *internet.* Esta ferramenta, embora segundo descrição da IBM, trata-se de uma ferramenta bem completa, mas não realiza verificação de segurança com base nos requisitos da norma ISO/IEC 15408.

O *Lapse,* por sua vez, é uma ferramenta sob licença General Public License (GPL), projetada para o auxilio na realização de auditorias de segurança em aplicações J2EE. A vantagem desta ferramenta é o fato dela ser gratuita, possibilitando sua utilização por quem necessitar de uma ferramenta para o auxilio nas verificações de segurança. A desvantagem é que o *Lapse* não utiliza um padrão oficial para verificação de conformidades e as verificações serem voltadas apenas para a web.

4 CONCLUSÕES

Com o crescimento cada vez mais rápido de tecnologias, a informatização de processos de todos os setores da economia e a conectividade, a segurança da informação nunca foi tão necessária. Empresas que realizam negócios em todo o Brasil através da internet, softwares de gestão, controle financeiro, operações bancárias na internet são alguns dos muitos motivos que geram preocupação com relação à segurança da informação e segurança no desenvolvimento de sistemas.

A padronização e criação de regras para verificação de conformidades de segurança após a criação da norma ISO/IEC 15408 tornou o processo de análise de requisitos de segurança de um sistema mais clara.

A realização de uma auditoria para verificar conformidades com base na norma, quando realizada de forma não automatizada, pode ser um processo um pouco tedioso e demorado, tornando a criação de testes que realizam a verificação semi-automatizada e agregar ao SCAN-CC, software de realização de auditoria em sistemas, é uma necessidade.

Neste trabalho foram desenvolvidos oito testes semi-automatizados, que realizam a verificação de conformidades segurança, levando em conta as regras de auditoria e segurança e identificação e autenticação descritas na norma. Houve uma grande dificuldade no desenvolvimento destes testes, na busca de realizá-los de forma mais genérica possível, mas como não há a possibilidade de verificar o código desenvolvido sem fazer uma análise mais profunda, optou-se então a utilização das interfaces para que durante o desenvolvimento já possam ser seguidos padrões de segurança implementando-as. Mesmo havendo esta dificuldade, este trabalho verifica de forma semi-automatizada 100% das regras descritas atendendo aos objetivos iniciais e 80% dos testes da família de auditoria e segurança e identificação e autenticação.

Mesmo com este trabalho completo, ainda existe a necessidade de implementação de mais testes para que atendam as outras regras de segurança da norma, como também a implementação de um WebService que permitirá à ferramenta analisar códigos implementados em outras linguagens.

4.1 EXTENSÕES

A seguir são apresentados alguns pontos que podem ser agregados ou melhorados no software:

a) implementar testes para mais requisitos da norma ISO/IEC 15408 seguindo uma determinada regra;
b) adaptar a ferramenta para sua utilização via web;
c) implementação de um WebService para dar possibilidade a ferramenta para verificação de sistemas não desenvolvidos em Java.

REFERÊNCIAS BIBLIOGRÁFICAS

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

CAMPOS, André L. N. **Sistema de segurança da informação**. Florianópolis: Visual Books, 2006. 180 p.

CYGNACON. **Specializing in Public Key Infrastructure (PKI) & Cryptography.** [S.l.], [2010?]. Disponível em: <http://www.cygnacom.com/labs/cc_assurance_index/CCinHTML/PART2/PART2CONTENTS.HTM>. Acesso em: 01 maio 2011.

COMMON CRITERIA. **The Common Criteria portal.** [S.l.][2009?]. Disponível em: <http://www.commoncriteriaportal.org/>. Acesso em: 21 jun. 2011.

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

IBM. **Rational appScan express edition.** [S.l.], [2010?]. Disponível em: <http://www-01.ibm.com/software/awdtools/appscan/express/>. Acesso em: 10 set. 2010.

LEOPOLDO, Marcus R. B. **Simple Object Access Protocol:** entendendo o Simple Object Access Protocol (SOAP). [S.l.], [2003]. Disponível em: <http://www.msdnbrasil.com.br/secure/sharepedia/arquivos/SOAP.pdf>. Acesso em: 12 set. 2010.

LIVSHITS, Benjamin. **LAPSE:** web application security scanner for Java. [S.l.], [2006]. Disponível em: <http://suif.stanford.edu/~livshits/work/lapse/index.html>. Acesso em: 10 set. 2010.

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

TRAPP, Dayana F. **Software para verificação de conformidade de sistemas à norma ISO/IEC 15408**. 2010. 66 f . Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.