CENTRO PAULA SOUZA

FACULDADE DE TECNOLOGIA DE AMERICANA

Curso De Análise e Desenvolvimento de Sistemas

Rafael Yukiharu Yamagawa

Benefícios do teste na segurança do *software*

Americana, SP

Dezembro / 2015

CENTRO PAULA SOUZA

FACULDADE DE TECNOLOGIA DE AMERICANA

Curso De Análise e Desenvolvimento de Sistemas

Rafael Yukiharu Yamagawa

Benefícios do teste na segurança do *software*

Trabalho de Conclusão de Curso desenvolvido em cumprimento à exigência curricular do Curso Análise e Desenvolvimento de Sistemas, sob a orientação do Prof. Me. Anderson Luiz Barbosa.

Área de concentração: Engenharia de *Software.*

Americana, SP

Dezembro / 2015

Yamagawa, Rafael Yukiharu
Y18b Benefícios do teste na segurança do software. / Rafael Yukiharu Yamagawa. – Americana: 2015.
54f.

Monografia (Graduação em Tecnologia em Análise e Desenvolvimento de Sistemas). - - Faculdade de Tecnologia de Americana – Centro Estadual de Educação Tecnológica Paula Souza.
Orientador: Prof. Me. Anderson Luiz Barbosa

1. Segurança em sistemas de informação I. Barbosa, Anderson Luiz II. Centro Estadual de Educação Tecnológica Paula Souza – Faculdade de Tecnologia de Americana.

CDU: 681.518.5

Rafael Yukiharu Yamagawa

Benefícios do teste na segurança do *software*

Trabalho de graduação apresentado como exigência parcial para obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas pelo CEETEPS/Faculdade de Tecnologia – FATEC/Americana.

Área de concentração: Engenharia de *Software*.

Americana, 11 de Dezembro de 2015

Banca Examinadora

Anderson Luiz Barbosa (Presidente)
Mestre
FATEC Americana

Benedito Aparecido Cruz (Membro)
Graduado
FATEC Americana

José Luís Zem (Membro)
Doutor
FATEC Americana

RESUMO

A segurança é muito discutido na atualidade e de extrema importância em qualquer produto ou serviço. Como forma de garanti-la no desenvolvimento de *software*, é necessário elaborar ótimos requisitos de segurança, realizar uma boa codificação e conhecer muito bem todas as tecnologias que serão utilizadas. Para facilitar a ação de minimizar as vulnerabilidades existentes e, deste modo, garantir que não ocorrerá diversas falhas na segurança, pode-se fazer uso dos testes, sendo estes processos importantes no desenvolvimento de um *software*, pois possuem a finalidade de localizar fraquezas. Como forma de validar os benefícios que os testes podem trazer para a segurança, primeiramente é definido o que é *software* seguro e o conceito de testes, além de explorar o funcionamento de cada técnica de teste utilizada. Após essa parte teórica, foi realizado um estudo de caso, em que foram executadas as técnicas vistas (teste estrutural, teste funcional e teste de segurança) em um sistema *web* desenvolvido utilizando-se o Django. Esta monografia apresenta toda essa parte explicativa teórica e a aplicação prática de todos estes conceitos.

Palavra chave: Teste de *software,* segurança, *software* seguro.

ABSTRACT

The security is very discussed nowadays and extremely important in any product or service. In order to guarantee it in software development, it is necessary to elaborate optimal security requirements, perform a good coding and know very well all the technologies that will be used. As a way to facilitate the action of minimize vulnerabilities and, that way, assuring that will not occur several security flaws, can make use of tests, which are important processes in software development, because they are intented to locate weakness. In order to validate the benefits that the tests can bring to security, first it is defined what is safe software and the concepts of tests, also, explore the operation of each test technique. After this theoretics, it was elaborate a case study, on which three techniques (structural test, behavioral test and security test) were performed on a web system developed using Django. This monograph features all this theoretics and the practical application of those concepts.

Key words: software test, security, safe software.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

CSRF – *Cross-Site Request Forgery*

CWE – *Common Weakness Enumeration*

GFC – Gráfico de Fluxo de Controle

HTTPS – *Hyper Text Transfer Protocol Secure*

IEC – *International Electrotechnical Commission*

IEEE – *Institute of Electrical and Electronics Engineers*

ISO – *International Organization for Standardization*

IT – *Information Technology*

OS – *Operational System*

OWASP – *Open Web Application Security Project*

SQL – *Structured Query Language*

TI – Tecnologia da Informação

URL – *Universal Resource Locator*

1 INTRODUÇÃO

A segurança é um assunto muito comentado nos dias atuais e um fator essencial em qualquer produto ou serviço. Como forma de exemplificar esta importância, pode-se vislumbrar a produção de um automóvel, em que é necessário pensar no bem estar dos funcionários e na capacidade do produto de garantir que os seus usuários não sofram qualquer tipo de dano.

Quando se trabalha com *softwares* as coisas não são diferentes. Ao desenvolver um sistema, necessita-se focar na segurança tanto das máquinas físicas como das informações. Deste modo, nota-se a necessidade de um planejamento bem feito de requisitos voltados para a segurança.

Visto a importância da segurança, agora pode-se conceituar o que é um *software* seguro. Considerando a informação como um bem valoroso, um *software* seguro pode ser definido como todo *software* que garanta “a confidencialidade, integridade, disponibilidade e demais aspectos da segurança das informações” (LYRA, 2008, p. 04), com o qual ele trabalha. Nota-se, também, a existência de três fatores que influenciam na segurança: o planejamento, a codificação e a tecnologia utilizada.

Como forma de minimizar futuras falhas de segurança, pode-se utilizar os testes para detectar as fraquezas existentes, pois o objetivo destes é localizar vulnerabilidades.

Os testes são de extrema importância em um processo de desenvolvimento, pois são eles que irão garantir a qualidade final no produto, detectando erros antes que um usuário o faça. Não obstante, os teste são negligenciados por diversas empresas, sendo tratado como um gasto descartável. Além das empresas, muitas instituições de ensino também desprezam o teste, não os incluindo na grade curricular ou as inserindo em outras matérias, onde serão vistas de forma superficial.

Esta falta de valor sobre o tema levou a elaboração desta monografia, que possui como objetivo principal demonstrar que o teste possui uma importância significativa. Deste modo, aplicá-lo na segurança, que é um tema muito discutido hoje, foi uma forma de comprovar de fato que o teste pode trazer benefícios para um *software*.

Assim sendo, serão tratados conceitos relacionados a segurança do *software*, ou seja, definição de *software* seguro e de segurança da informação, vulnerabilidades, normas ISOs voltadas para a segurança e os três fatores influenciadores. Também serão vistos o conceito de teste e de três técnicas especificas, o teste funcional, o teste estrutural e o teste de segurança. Para finalizar, os testes serão aplicados na prática em um sistema *web* real que foi desenvolvido em Django.

Para melhor demonstrar todo esse conteúdo, a monografia foi dividida da seguinte maneira: no próximo capitulo será tratado a segurança do *software*, sendo dividido em quatro subcapítulos, segurança da informação, vulnerabilidades, normas e fatores. O capitulo 3 irá falar sobre a definição de teste e sobre as três técnicas (funcional, estrutural e de segurança). No capítulo 4 será feito o estudo de caso, em que foi planejado os requisitos de segurança, os casos de testes, e onde foi executado os testes, sendo informado os resultados obtidos e uma avaliação final em cima dos requisitos. Para finalizar, tem-se as considerações finais.

2 SEGURANÇA DE *SOFTWARE*

Na atualidade, o termo segurança é muito discutido, sendo um fator importante em muitas áreas do mercado. Desde alimentos à automóveis, produtos a serviços, quando algo é classificado como seguro, este ganha uma importância e agrega valores pela sua credibilidade. Afinal, você escolheria um carro considerado seguro ou um que apresente diversas falhas?

Em relação a *softwares,* as coisas não são diferentes. *Software* seguro agrega muito mais valor e confiança, sendo adquirido por um maior número de clientes, ganhando maior credibilidade no mercado.

No entanto, antes de definir o que é um *software* seguro, é necessário entender o significado de segurança. Segundo o dicionário da língua portuguesa, segurança significa “Certeza, firmeza, convicção” (FERREIRA, 2010, p. 1906). Sendo assim, pode-se definir como seguro tudo aquilo que demonstra, certeza, proteção e confiança.

Atrelado a esta definição, segurança de *Software* mostra-se bem mais complexa do que aparenta, pois consiste em um projeto extenso, que envolve desde o levantamento de requisitos até a manutenção. Esta monografia tem como principal foco a parte de desenvolvimento do *software*, para ser mais preciso, apenas a parte lógica, visto que esta é apenas uma parcela do processo. Ao desenvolver um sistema, estão inclusas a segurança física (*backup*, acesso autorizado ao código-fonte, controle de versões, entre outros) e a segurança lógica (tecnologia utilizada, usuário e senha, permissões, etc.).

Deste modo, esclarecido o que é segurança e como ela funciona em um *software,* pode-se chegar a ideia de que *software* seguro é aquele que garante a segurança das informações com as quais trabalha. Para melhor entendimento será necessário ver o conceito de segurança da informação.

2.1 SEGURANÇA DA INFORMAÇÃO

Atualmente, vive-se em uma sociedade completamente virtualizada, na qual a computação exerce papel essencial em qualquer organização do mundo, não possuindo mais um papel secundário de caráter auxiliar. Nesse sentido, a informação, que está intimamente ligada a função chave do mecanismo computacional, pois este é responsável por assegurá-la, se tornou um bem extremamente valioso.

Portanto, com a intenção de prevenir a perda ou o roubo deste bem tão valioso, surge a segurança da informação, que nada mais é do que “ações para garantir a confidencialidade, integridade, disponibilidade e demais aspectos da segurança das informações dentro das necessidades do cliente” (LYRA, 2008, p. 04). Confidencialidade, integridade e disponibilidade são aqueles considerados os princípios da segurança da informação. No entanto, os demais aspectos, que são a autenticação, a legalidade e a auditoria, são igualmente essenciais, tendo que, junto com os três princípios, compõem um todo bem sincronizado em torno de uma única causa: a segurança informacional.

2.1.1 Confidenciabilidade

Primeiramente, tem-se confidencialidade como a “capacidade de um sistema de permitir que alguns usuários acessem determinadas informações ao mesmo tempo em que impede que outros, não autorizados, a vejam” (LYRA, 2008, p. 03). Com isso, por estar restrita a um número pré-determinado de indivíduos, garante-se maior segurança e controle da informação.

Tal segurança e controle consistem no fato de que a informação, sendo confidencial, estará protegida de pessoas maliciosas, capazes de usá-la indevidamente.

De maneira a exemplificar a importância deste princípio da segurança, pode-se pensar em bancos, onde existem os dados sobre clientes e contas bancárias. Todos estes dados possuem extremo valor e, por motivos bem claros, devem ser confidenciais a um número de pessoas qualificadas (todos os funcionários do banco). Portanto, na ausência da confidencialidade informacional, inúmeros clientes poderiam ser prejudicados por outros indivíduos que, nesse caso, teriam acesso à conta bancária destas pessoas e, desse modo, estariam aptos a fazerem uso indiscriminado e indevido de seus capitais armazenados. Sendo assim, além do cliente a instituição também é prejudicada em relação a sua credibilidade no mercado.

2.1.2 Integridade

Em relação a integridade de uma informação, pode-se dizer que ela “lida com a validade e a precisão de dados” (KIM; SOLOMON, 2014, p. 10). De modo mais específico, deseja-se indicar que ela não sofreu nenhum tipo de alteração. Por conseguinte, ao se falar de uma perda na integridade informacional, afirma-se também que sua confiança foi danificada.

De modo prático, toda informação destituída de integridade afetará negativamente a organização a que pertence, já que ela pode estar diferente do seu original. Em relação ao agente desta modificação, pode-se dizer que ele tenha sido um elemento externo mal-intencionado e para evitá-lo, a segurança se mostra essencial.

Como forma de demonstrar o papel da integridade de forma prática tem-se o seguinte exemplo: uma empresa aérea recebe informação de suas aeronaves em tempo real, fazendo com que a torre de controle consiga guiá-las corretamente e evitar grandes tragédias. Pense, então, que uma destas informações foi corrompida e, desta forma, o aviso da torre de controle para a aeronave não condiz com a situação atual. Isto pode ocasionar uma mudança imprevista na rota, causando, no

cenário bom, o atraso do voo e , no cenário ruim, um acidente aéreo.

2.1.3 Disponibilidade

A informação deve sempre estar disponível para que as pessoas consigam acessá-la. Caso isto não aconteça, pode ocorrer uma perda significativa para as organizações, visto que não conseguir dispor de uma informação valiosa quando necessário é um cenário muito ruim e prejudicial.

Tem-se o seguinte cenário. Uma grande empresa mantém diariamente contato com diversos clientes ao dia, vendendo um enorme número de produtos. Caso os dados dos clientes, do estoque e dos produtos se tornem indisponíveis para os vendedores por um tempo considerável, estes não tem como efetuarem as vendas e, assim, a empresa perderá dinheiro e confiança.

2.1.4 Autenticação

Todo usuário que tiver acesso as informações deve possuir uma autenticação para, com isso, mostrar que ele é de fato quem diz ser.

Como foi dito, em confidencialidade, onde a informação só deve ser acessada por quem tiver permissão para fazer isso, para conhecer se uma pessoa possui ou não autorização de acesso é que existe a autenticação. Ao autenticar-se em um sistema, um indivíduo terá acesso apenas aquilo que, previamente, foi-se declarado que ele teria, não podendo obter dados fora de seu poder.

Além de garantir poder e controlar a permissão dos usuários, a autenticação também ajudará a verificar se todos estão realizando aquilo que deveriam realizar, impedindo possíveis fraudes, ou má utilização daquilo que está disponível.

Um exemplo de autenticação seria um *login* de usuário e senha presente em um sistema de vendas, em que os funcionários necessitariam de uma conta para poderem acessarem todas as informações de vendas, dos clientes e dos produtos.

2.1.5 Auditoria

Ao auditar um sistema, está sendo verificado todas as ações realizadas e os usuários envolvidos nestas ações para, desta forma, ter-se um controle em todas as informações e em tudo o que acontece dentro deste sistema.

Com a auditoria é muito fácil encontrar fraudes, ataques, ou qualquer tipo de ação maliciosa que pode prejudicar as informações, e os indivíduos por trás de tudo.

2.1.6 Legalidade

Na sociedade atual existem muitas leis e normas que devem ser seguidas a risca pelas pessoas e pelas organizações. É necessário respeitar toda essa legislação, por isso que legalidade é um aspecto da segurança da informação, pois para que as informações sejam válidas e aceitadas, elas devem obedecer normas ou regras prontas.

2.2 VULNERABILIDADES

2.2.1 Conceito

Vulnerabilidades podem ser definidas como as fraquezas, pontos fracos, de um *software,* estando sempre presentes na realidade, visto que algo considerado perfeito e excluso de fragilidades só pode ser avistado em um mundo utópico.

Com isso, então, todos os *softwares* são inseguros por natureza? Como todos

eles sempre apresentarão vulnerabilidades, não há como confiar em nenhum deles? Estas questões fazem todo sentido, contudo para verificar o nível de segurança não há como contar apenas com a variável conhecida como vulnerabilidade, existe outro fator que deve ser levado em conta, as ameaças.

2.2.2 Ameaças

Segundo KIM e SOLOMON, “ameaça é qualquer ação que possa danificar um bem” (2014, p. 05). Com isso, pode-se notar que fenômenos naturais são considerados ameaças, pois estes podem prejudicar de forma física a segurança da informação como, por exemplo, danificando servidores de armazenamento.

Por outro lado, quando há a existência de um agente, tanto interno quanto externo, que busca atingir determinado sistema com a intenção de acessar suas informações ou danificá-las, estes são considerados ameaças humanas.

Porém, para adquirir tal objetivo, estes indivíduos necessitam de uma brecha, por este motivo que eles rastreiam e fazem uso das vulnerabilidades do sistema, sendo que estes, como foi visto, sempre estarão presentes.

2.2.3 Falhas de segurança

No momento que uma ameaça obtêm sucesso em utilizar uma vulnerabilidade para invadir ou quebrar um *software*, tem-se a falha de segurança.

Tendo em vista o conceito de falha de segurança, há como supor o nível de segurança de um sistema fazendo uso da quantidade de vulnerabilidades e o número de ameaças que este apresenta, sendo o último apenas uma hipótese, pois não há como controlar um agente externo.

Portanto, pode-se dizer que um sistema possui alto nível de insegurança quando o número de vulnerabilidades e de ameaças que este apresentar forem elevados. Por outro lado, quando a quantidade de ambas forem baixas, então há um alto nível de segurança.

Agora, quando um *software* possuir muitas vulnerabilidades e poucas ameaças, ou vice-versa, pode-se pensar da seguinte maneira: se um sistema possui muitas fraquezas, porém não há ameaças para fazerem uso destas, este seria um sistema seguro, ou, se há muitas ameaças, mas quase nenhum ponto fraco para atacar, assim sendo, o nível de segurança pode ser considerado bom.

2.2.4 Vulnerabilidades mais perigosas

Como pode ser observado no site da CWE[1], existem 25 vulnerabilidades consideradas as mais perigosas (dados de 2011), dado que elas são fáceis de se encontrar e de se explorar, permitindo que agentes possam controlar o *software*, roubar informações, ou impedir o funcionamento adequado. Nesta lista pode-se encontrar diversas fraquezas que são muito conhecidas como o *SQL Injection*, o *Classic Buffer Overflow* e o *Cross-site Scripting*. Abaixo está uma tabela contendo a lista completa:

Tabela *1:* 25 vulnerabilidades mais perigosas

Rank	Score	ID	Name
[1]	93.8	CWE-89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection'')
[2]	83.3	CWE-78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection'')
[3]	79.0	CWE-120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow'')
[4]	77.7	CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting'')
[5]	76.9	CWE-306	Missing Authentication for Critical Function

1 https://cwe.mitre.org/

Rank	Score	ID	Name
[6]	76.8	CWE-862	Missing Authorization
[7]	75.0	CWE-798	Use of Hard-coded Credentials
[8]	75.0	CWE-311	Missing Encryption of Sensitive Data
[9]	74.0	CWE-434	Unrestricted Upload of File with Dangerous Type
[10]	73.8	CWE-807	Reliance on Untrusted Inputs in a Security Decision
[11]	73.1	CWE-250	Execution with Unnecessary Privileges
[12]	70.1	CWE-352	Cross-Site Request Forgery (CSRF)
[13]	69.3	CWE-22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal'')
[14]	68.5	CWE-494	Download of Code Without Integrity Check
[15]	67.8	CWE-863	Incorrect Authorization
[16]	66.0	CWE-829	Inclusion of Functionality from Untrusted Control Sphere
[17]	65.5	CWE-732	Incorrect Permission Assignment for Critical Resource
[18]	64.6	CWE-676	Use of Potentially Dangerous Function
[19]	64.1	CWE-327	Use of a Broken or Risky Cryptographic Algorithm
[20]	62.4	CWE-131	Incorrect Calculation of Buffer Size
[21]	61.5	CWE-307	Improper Restriction of Excessive Authentication Attempts
[22]	61.1	CWE-601	URL Redirection to Untrusted Site ('Open Redirect'')
[23]	61.0	CWE-134	Uncontrolled Format String
[24]	60.3	CWE-190	Integer Overflow or Wraparound
[25]	59.9	CWE-759	Use of a One-Way Hash without a Salt

Fonte: https://cwe.mitre.org/top25/index.html

2.3 NORMAS

Existem normas específicas para a segurança da informação, muitas delas são voltadas para técnicas de segurança enquanto outras focam na qualidade do produto.

Quase todas as normas existentes fazem parte da ISO, por isso são aceitas em grande parte do mundo. Além da ISO existem outras organizações, como o IEEE, que possuem padrões voltadas para a segurança, mesmo não sendo tão amplas em comparação com a primeira.

Além de instituições que fornecem normas e regulamentos, há aquelas que apenas orientam e proporcionam serviços de consultoria como a OWASP, uma fundação internacional.

2.3.1 ISO/IEC 27001:2013 – *Information technology -- Security techniques -- Information security management systems – Requirements*

Esta é uma norma internacional de gestão de segurança da informação. Ela possui como função principal especificar “os requerimentos para estabelecer, implementar, manter e melhorar continuamente um sistema de gerenciamento de segurança da informação dentro do contexto da organização” (ISO/IEC, 2013, Trad. própria).

2.3.2 ISO/IEC 27002:2013 – *Information technology -- Security techniques -- Code of practice for information security controls*

Esta norma, anteriormente conhecida como norma ISO 17799, complementa a ISO/IEC 27001:2013 providenciando “orientações para normas de segurança da informação organizacionais e práticas de gestão de segurança da informação”

(ISO/IEC, 2013, Trad. própria). Ela é designada para as organizações que pretendem implementar controles de segurança da informação e desenvolver diretrizes próprias de gestão de segurança.

2.3.3 ISO/IEC 27005:2011 – *Information technology -- Security techniques -- Information security risk management*

Esta norma possui a finalidade de prover boas práticas para gerenciar riscos de segurança da informação. É necessário conhecer todos os modelos e conceitos das normas ISO/IEC 27001 e ISO/IEC 27002, sendo designado “para auxiliar a implementação satisfatória da segurança da informação baseada em uma abordagem de gerenciamento de riscos” (ISO/IEC, 2013, Trad. própria).

2.3.4 ISO/IEC 15408 – *Information technology -- Security techniques -- Evaluation criteria for IT security*

Esta norma, também chamada de *Common Criteria*, é conhecida por ser uma referência universal para todos aqueles que lidam com a segurança da informação. Sua função é atestar que o software implemente um conjunto de funcionalidades de segurança.

Ela é dividida em três partes:

- ISO/IEC 15408-1:2009 – *Information technology -- Security techniques -- Evaluation criteria for IT security - - Part 1: Introduction and general model*

- ISO/IEC 15408-2:2008 – *Information technology -- Security techniques -- Evaluation criteria for IT security -- Part 2: Security functional components*

- ISO/IEC 15408-3:2008 – *Information technology -- Security techniques -- Evaluation criteria for IT security -- Part 3: Security assurance components*

A primeira parte da norma serve como uma introdução estabelecendo “os conceitos e princípios de avaliação de segurança de TI em geral e especifica o modelo geral de avaliação dada por várias partes do ISO” (ISO/IEC, 2009, Trad. própria). A segunda parte da ISO/IEC 15408 já possui a função de definir “o conteúdo e apresentação dos requisitos funcionais de segurança para ser avaliado em uma avaliação de segurança” (ISO/IEC, 2008, Trad. própria), contendo uma lista ampla de componentes funcionais pré-definidos de segurança que satisfará as necessidades mais comuns do mercado. A terceira, e última, parte da norma ISO/IEC 15408 busca definir “os requisitos de garantia dos critérios de avaliação” (ISO/IEC, 2008, própria), completando, assim, o objetivo geral da norma que é a avaliação dos métodos de segurança aplicados no software.

2.4 FATORES

Durante o desenvolvimento de um *software* existem três fatores que podem influenciar, de forma negativamente ou positivamente, a segurança deste, tudo depende da forma como será feito o use deles.

2.4.1 Planejamento de requisitos de segurança

Segundo LYRA, “como segurança não é um parâmetro único, precisamos saber a necessidade do cliente e do usuário do sistema em cada aspecto da segurança” (2008, p. 75). Assim sendo, é necessário planejar uma série de requisitos funcionais voltados apenas para a segurança do *software*, deste modo será minimizado as vulnerabilidades que este apresentar no futuro.

2.4.2 Codificação

A codificação de um sistema é a parte principal de um projeto, por este motivo é necessário que seja bem planejado e bem efetuado. Quando um código é escrito focando-se na qualidade dele, diminui as vulnerabilidades que ele pode apresentar e aumenta a facilidade de corrigir eventuais erros, pois será facilmente entendido por outros programadores.

Algumas das recomendações de como produzir um código mais seguro e de qualidade são:

- Criar funções que verifiquem os dados de entrada, evitando perda de controle ou falhas como, por exemplo, o *buffer overflow*[2];
- Utilizar apenas funções seguras;
- Testar o retorno de funções;
- Documentar o código;
- Tratar a entrada de dados;
- Adotar uma politica de versionamento de código;
- Usar componentes e bibliotecas confiáveis;
- Não armazenar senhas dentro do código;
- Operar com privilégios necessários;
- Tratar todas as entradas do sistema como não seguras.

2 Quando um programa recebe mais dados do que pode armazenar no *buffer.*

2.4.3 Tecnologias

Como foi dito anteriormente, todos os *softwares* possuem vulnerabilidades, sendo elas ainda não encontradas ou de conhecimento público, desse modo, pode-se concluir que ao se utilizar de uma tecnologia para o desenvolvimento de um sistema, está sendo incorporado todas as fraquezas de um no outro.

Por conseguinte, é necessário possuir discernimento sobre todas as fragilidades notórias de uma tecnologia antes de fazer uso desta para, deste modo, conseguir suprimi-las por outros meios durante a codificação do *software*.

3 TESTE DE *SOFTWARE*

Neste capítulo será visto a definição de teste de *software* e os benefícios que ele traz. Além disso, ele mostrará três técnicas de testes existentes, funcional, estrutural e de segurança, e suas finalidades, além de, relacioná-las com os três fatores que influenciam a segurança do *software* como foi visto no capítulo anterior.

3.1 DEFINIÇÃO E BENEFÍCIOS

Segundo RIOS e MOREIRA, “teste de software é o processo que visa a sua execução de forma controlada, com o objetivo de avaliar o seu comportamento baseado no que foi especificado” (2006, p. 08), desta forma é possível detectar a presença de diversos defeitos.

Tendo definido o que é o teste, agora se deve elucidar o seu intento. DELAMARO, MALDONADO E JINO esclarecem que “o objetivo de teste não é, como podem pensar muitos, mostrar que um programa está correto […] o objetivo é mostrar a presença de defeitos caso eles existam” (2007, p. 06).

Muitos podem não pensar assim, contudo o teste é um processo de extrema importância no desenvolvimento de *software*. Mesmo que os próprios programadores realizem testes naquilo que estão desenvolvendo, estes não passam de apenas testes unitários simples, servindo apenas para verificar a funcionalidade de um trecho de código.

Pode-se notar que diversos *softwares* são liberados para utilização contendo vários erros. Isto acontece, pois as empresas consideraram os testes realizados pelos próprios programadores como aceitáveis e ignoram a importância de um processo focado apenas em testes exaustivos com a intenção de ocasionar erros no *software*.

Mesmo possuindo um custo maior para criar todo um setor e uma politica voltada para o teste, no futuro haverá uma maior economia, pois ao aplicar técnicas de testes, as falhas do sistema serão minimizadas, diminuindo reclamações e manutenções caras, além de aumentar a confiabilidade da empresa.

Tendo em vista que o intuito do teste é encontrar falhas, pode-se utilizá-lo para a finalidade de garantir a maior segurança em um *software*, pois, com procedimentos de testes bem planejados e executados, consegue-se detectar diversas vulnerabilidades tanto no *software* como nas tecnologias utilizadas.

3.2 TÉCNICAS

Existem diversas técnicas de teste de *software*, cada uma possui uma finalidade própria. Nesta monografia será trabalhado técnicas voltadas para a detecção de vulnerabilidades em um *software*, levando em conta aquilo que foi visto no capítulo anterior. As técnicas que serão mencionadas são: teste de Caixa Preta, teste de Caixa Branca e teste de segurança.

3.2.1 Teste de Caixa Preta

Também conhecido como teste funcional, é uma técnica utilizada para verificar se as saídas geradas estão de acordo com os objetivos especificados, para isso são fornecidas diversos dados de entrada no sistema.

Ele é conhecido como Caixa Preta, pois não se consegue observar a estrutura do código, visando observar o sistema “em uma ótica externa ou do usuário, sem se basear em qualquer conhecimento do código e da lógica interna do componente testado” (RIOS; MOREIRA, 2006, p. 13), assim sendo, só é verificado se os dados fornecidos e os dados emitidos são exatamente o esperado.

Em princípio, o teste funcional pode detectar todos os defeitos, submetendo

> o programa ou sistema a todas as possíveis entradas […]. No entanto, o domínio de entrada pode ser infinito ou muito grande, de modo a tornar o tempo da atividade de teste inviável (DELAMARO et. al., 2007, p. 09).

Deste modo, pode-se abordar o teste funcional de dois modos distintos, o primeiro, conhecido como teste randômico, consiste em inserir uma grande quantidade de dados aleatoriamente e, assim, eliminar várias possibilidades possíveis de defeito.

O segundo, chamado de teste de partição, já busca limitar todas as possibilidades em grupos distintos e, após isto, testar apenas um dado de cada um destes grupos.

Como o teste de Caixa Preta apenas se baseia em especificações, sua qualidade irá se sustentar no quão bom foram planejados os requisitos. Basicamente, ao realizar procedimentos de testes funcionais, o resultado obtido irá ser comparado com o resultado esperado pelo cliente, o qual pormenorizou tudo nos requisitos.

Visto que existem duas abordagens em que se pode conduzir as atividades de testes, existem diversos critérios funcionais que são aplicados na segunda abordagem (teste de partição). Os mais conhecidos são Particionamento de Equivalência e Análise do Valor Limite, existindo outros além desses, todavia, não serão abordados aqui.

- Particionamento de Equivalência

Como dito anteriormente, é inviável aplicar um teste exaustivo, por este motivo muitos profissionais restringem todas as possíveis entradas em diversos conjuntos, não obstante, estes conjuntos devem possuir uma alta possibilidade de encontrar defeitos.

Para limitar a infinidade de prováveis entradas, este critério de teste funcional

busca dividir o domínio de dados em conjuntos baseados em classes de equivalência, ou seja, a divisão será realizada de forma que dados tratados da mesma maneira pelo programa façam parte de um mesmo grupo, como, por exemplo, os números 5 e 10 em que ambos são podem fazer parte de um conjunto que contenha números inteiros e maiores do que 0. Tendo isto em mente, pode-se considerar que qualquer dado pertencente a uma classe de equivalência pode ser o representante do grupo, pois esperasse que qualquer resultado encontrado com este representante seja igual com outras entradas desta mesma classe.

Para aplicar este critério de teste funcional é necessário que primeiramente sejam delimitadas as classes de equivalência. Após definir as classes de equivalência é preciso elaborar os casos de testes, estes detalharão as entradas escolhidas, podendo ser uma para cada classe, e então definir a saída que é esperada para cada entrada. Depois de dividir os conjuntos e planejar os casos de testes é só executá-los e comparar o resultado obtido com a saída esperada, sendo que, aqueles que não baterem podem demonstrar algum defeito presente, tanto no *software* criado, código ou tecnologias utilizadas, quanto naquilo que foi planejado.

Como forma de exemplificar o critério tem-se o seguinte ambiente: um sistema realiza cálculos de divisão e recebe duas entradas, uma é o dividendo e a outra o divisor. Agora é necessário a divisão dos conjuntos, primeiro será criado um contendo apenas o número 0, visto que, não existe divisão por zero. Assim sendo, neste caso, tem-se um grupo considerado inválido, pois tem-se o conhecimento de que deve ocorrer um erro ao utilizá-lo como entrada.

Além deste conjunto, pode-se definir outros conjuntos que tendem ao erro, como, por exemplo, os caracteres e os caracteres especiais. Com isso se consegue elaborar a seguinte tabela.

Tabela 2: Conjunto de dados

Conjunto	Conteúdo
Nulo	0

Caracteres	a, b, c, d, e, f, g, h, i, j, k ...
Caracteres Especiais	! , . ; / ? [ ] _ - + = ( ) …

Fonte: Própria

Agora que foram definidos os primeiros grupos, pode-se criar os demais. Para isso é necessário apenas pensar nos outros números, tendo os inteiros maiores que zero e os inteiros menores que zero. Não será utilizado outros conjuntos, pois este é um simples exemplo para melhor compreensão do critério.

Tabela 3: Conjunto de dados completo

Conjunto	Conteúdo
Nulo	0
Caracteres	a, b, c, d, e, f, g, h, i, j, k ...
Caracteres Especiais	! , . ; / ? [ ] _ - + = ( ) …
Inteiros Positivos	1 2 3 4 5 6 7 8 9 10 11 ...
Inteiros Negativos	-1 -2 -3 -4 -5 -6 -7 -8 -9 -10 ...

Fonte: Própria

Após definir os conjuntos é necessário elaborar os casos de testes, bastando utilizar um dado de cada conjunto como entrada e definir uma saída esperada. No caso deste exemplo, no qual possui duas entradas, tem-se que criar diferentes sequências para, assim, todos os dados sejam testados em ambas as entradas com todas as combinações possíveis.

Tabela 4: Caso de teste

Caso de teste		
Código	**Entradas**	**Saída esperada**
1	0 / 0	Erro
2	0 / a	Erro
3	0 / !	Erro

4	0 / 1	0
5	0 / -1	0
6	a / 0	Erro
7	a / a	Erro
8	a / !	Erro
9	a / 1	Erro
10	a / -1	Erro
11	! / 0	Erro
12	! / a	Erro
13	! / !	Erro
14	! / 1	Erro
15	! / -1	Erro
16	1 / 0	Erro
17	1 / a	Erro
18	1 / !	Erro
19	1 / 1	1
20	1 / -1	-1
21	-1 / 0	Erro
22	-1 / a	Erro
23	-1 / !	Erro
24	-1 / 1	-1
25	-1 / -1	1

Fonte: Própria

Após definir o caso de teste, é necessário executá-lo e, então, combinar o resultado obtido com o esperado. Ao ocorrer algo diferente do planejado encontra-se um defeito no sistema.

- Análise de Valor Limite

Este critério é utilizado junto com o critério anterior, portanto, ele também fará a delimitação de todas as possíveis entradas, separando-as em conjuntos baseados na equivalência dos dados.

Em contrapartida, ele se difere do Particionamento de Equivalência no seguinte ponto, ele não se utiliza de um dado aleatoriamente apanhado de um conjunto, ele se utiliza dos limites dos conjuntos como entrada.

Para clarear, em vez de pegar um dado qualquer, ele faz uso dos dados que delimitam os grupos, como, por exemplo, em um conjunto de números maiores que 0 e menores que 10 será utilizado tanto o 1 e o 9 quanto o 0 e o 10. No primeiro caso são dados considerados válidos, que devem funcionar, e no segundo caso são dados que podem não funcionarem como o esperado deste grupo, sendo declarados inválidos.

3.2.2 Teste de Caixa Branca

Também conhecido como teste estrutural, esta técnica visa avaliar o próprio código fonte, a lógica utilizada, as configurações e outros elementos técnicos. Neste caso tem-se acesso à codificação e uma visão mais técnica do sistema.

Segundo DELAMARO, MALDONADO e JINO, “os caminhos lógicos do software são testados, fornecendo-se casos de testes que põem à prova tanto conjuntos específicos de condições e/ou laços bem como pares de definições e uso de variáveis” (2007, p. 47).

Consequentemente, esta técnica será mais utilizada pelos próprios desenvolvedores do *software* e, isto, pode gerar benefícios, pois haverá uma visão diferenciada em relação ao teste funcional, desta forma, tem-se como encontrar uma outra gama de efeitos, impossíveis de serem detectadas se vista do ponto de vista externo do usuário.

Todavia, o teste estrutural possui limites, apenas é necessário observar o seu objetivo. Como ele, basicamente, busca verificar se um trecho de código ou uma função faz exatamente o que foi planejado para ser feito, não é considerado, por

exemplo, se todos os caminhos contendo a função serão executados ou, ainda, se uma função será executada de forma igual em toda parte do programa que chamá-la.

Com isso em mente, nota-se que o teste de Caixa Branca sozinho não consegue identificar problemas no sistema propriamente dito, por este motivo criar todo um processo de testes utilizando apenas esta técnica não é recomendado, deve-se utilizá-lo em conjunto com outras técnicas voltadas para a funcionalidade do sistema completo, como, por exemplo, o teste funcional.

O teste estrutural faz uso de uma representação conhecida como GFC (Gráfico de Fluxo de Controle), no qual um programa é “decomposto em um conjunto de blocos disjuntos de comandos” (DELAMARO et. al., 2007, p. 51).

Assim sendo, pode-se considerar os blocos de comandos como sendo nós onde, um destes, é inicio do gráfico e, outro, o fim. Desde o primeiro até o último bloco podem existir outros e todos eles seguem um fluxo, sendo dois deles ligados por uma linha, ou arcos.

3.2.3 Teste de Segurança

Os testes de segurança possuem a finalidade de avaliarem “a capacidade de proteção do software contra acesso interno ou externo não autorizado” (RIOS; MOREIRA, 2006, p. 16). Esta técnica faz uso de força bruta para encontrar vulnerabilidades no *software*, ou seja, são testadas diversas maneiras de acessar os dados protegidos que o sistema ou programa possui.

Basicamente, esta técnica de teste busca precaver o *software* contra ataques maliciosos, garantindo uma maior segurança contra aqueles mais conhecidos e perigosos, e desta forma ele prepara medidas para caso aconteça um destes ataques. Desta forma é agregado maior valor ao sistema ou programa, garantindo

maior confiança ao cliente.

Muito diferente do teste funcional, o teste de segurança não trabalha com o que é de conhecimento mútuo ou o que já é bem definido, como, por exemplo, a entrada de caracteres em um campo texto onde deve ser feito diversas verificações. Pelo contrário, esta técnica que é mais voltada para a segurança da informação trabalha com comportamentos estranhos que muitas vezes não se conhecem o motivo ou gatilho para tal acontecimento.

Por estes motivos, o teste de segurança consegue detectar defeitos considerados fatais, porém minoria em relação aos outros, em um *software*. Contudo isto torna difícil o planejamento de um processo de teste que utilize esta técnica.

3.2.4 As técnicas de teste e a segurança do *software*

Como visto no capítulo anterior, todo *software* possui vulnerabilidades, o que pode acontecer é de algumas delas não serem conhecidas ainda. Ao relacionar isto com o objetivo dos testes (encontrar vulnerabilidades), não é de se estranhar ao se utilizar de técnicas de teste para minimizar problemas de segurança em um sistema.

Desta forma, as técnicas vistas neste capítulo possuem uma aplicação na detecção de vulnerabilidades na segurança da informação em um *software* já desenvolvido ou em fase de desenvolvimento.

Começando pelo teste estrutural, que possui como função verificar o código fonte e detectar se uma determinada função está realmente fazendo o que deveria, há uma boa chance de encontrar vulnerabilidades referentes a codificação, um dos fatores que podem afetar a segurança.

Um exemplo é detectar se a função de validação de usuário e senha está, de fato, verificando o banco de dados e comparando aquilo que foi digitado com o que

está guardado e permitindo o acesso de pessoas realmente autorizadas a isso. Tudo isto sem expor nenhum tipo de brecha para, por exemplo, um *SQL Injection.*

Já o teste funcional pode ser utilizado como benefício para a segurança da seguinte maneira, ao desenvolver um *software* é necessário o planejamento dos requisitos do cliente. Além deste requisitos voltados para a aceitação final, é vital a elaboração dos requisitos de segurança, que será voltada para a garantia da confiabilidade, integridade e disponibilidade de todas as informações com o qual este *software* trabalhará. Tendo em vista que o teste funcional verifica se o resultado obtido está de acordo com os requisitos elaborados, esta técnica permite verificar se o sistema ou programa satisfaz estes requisitos previamente planejados focados na segurança, acobertando o planejamento, um dos fatores de segurança.

Por último, o teste de segurança, por se tratar de uma técnica que é difícil estipular o que será utilizado como entrada, buscará vulnerabilidades anômalas. Assim sendo, ele irá expôr falhas que não foram encontradas pelas duas técnicas anteriores.

4 ESTUDO DE CASO

Visto todo o conceito teórico de segurança e de testes, neste capitulo será mostrado uma aplicação prática em um estudo de caso realizado sobre um sistema funcional. Aqui será mostrado o planejamento de requisitos e dos casos de testes e os resultados obtidos após a execução dos testes.

4.1 INTRODUÇÃO

O sistema *web* de inscrições para a III Escola de Primavera dos Intérpretes do Brasil, evento que ocorreu na Unicamp nos dias 21 à 23 de Outubro, contendo seminários na área de economia, foi desenvolvido em Python utilizando o *framework web* Django em conjunto com o *Bootstrap.*

Este sistema é dividido em três áreas, onde existe uma parte voltada para a inscrição, sendo liberada para o público geral, uma área de administração, restrita apenas ao administrador, e uma área de relatórios, que pode ser acessada por todos os funcionários envolvidos com o evento.

A área pública do sistema, denominado de *spring* pelo seu desenvolvedor, possui um formulário, em que os usuários poderão efetuar o cadastro no evento, além de informações relacionados aos seminários que ocorrerão durante os três dias. Nesta parte o indivíduo que deseja participar, preenche o formulário e então uma mensagem contendo uma chave é automaticamente enviada para o *e-mail* cadastrado, após isso é necessário que a pessoa entre no *site* novamente e, em uma parte específica do sistema, insira o CPF, previamente cadastrado, e a chave recebida.

A área responsável por gerar relatórios referentes ao evento é conhecido como Gelato, aqui é onde o pessoal responsável consegue acessar detalhes sobre os participantes e sobre os seminários. No Gelato não há como alterar dados

cadastrados ou inserir novos, apenas observar aquilo que já existe, pois grande parte das pessoas que acessam esta área do sistema não possuem domínio em computação e não fazem parte do grupo de desenvolvedores.

O administrativo necessita de *login* e, dentro dele, pode ser criado contas de usuários para acesso tanto nesta área como no Gelato (área que gera relatórios). É possível também observar detalhes dos participantes inscritos e dos seminários cadastrados, além de, realizar qualquer tipo de alteração.

Para a elaboração deste estudo de caso houve permissão por parte do desenvolvedor para acesso ao sistema como um todo, assim sendo, é possível executar testes tendo em vista tanto a parte dos usuários como dos responsáveis pelo evento e pela administração.

4.2 REQUISITOS DE SEGURANÇA

Os requisitos listados abaixo são focados na segurança do sistema. Eles foram elaborados tendo como objetivo garantir os três princípios da Segurança da Informação (confidencialidade, integridade e disponibilidade).

Pode-se notar que muitos destes requisitos são voltadas para as contas de usuários. Isso ocorre devido ao fato do sistema em estudo possuir tanto uma área pública quanto uma área de acesso restrito.

Tabela 5: Requisitos de segurança

ID	Requisito
RS1	Autenticar usuário e senha para acesso no administrativo e no Gelato.
RS2	Criar permissões de acesso, impedindo que um usuário faça algo que não possui autoridade.
RS3	Impedir a criação de contas com o campo 'Usuário' vazio e com o campo 'Senha' sendo muito simples.
RS4	Excluir contas de usuários que não acessam o sistema por mais de uma

	semana.
RS5	Solicitar usuário e senha novamente caso inativo por mais 10 minutos.
RS6	Campos de senha devem ser mascarados.
RS7	Utilizar o protocolo HTTPS.
RS8	Gravar ações que os usuários realizam no administrativo, como, por exemplo, excluir uma informação importante.
RS9	Senhas ou outros dados sensíveis não devem ser armazenadas no código fonte.
RS10	Ao cadastrar uma conta, gerar uma senha aleatória que deve ser alterada pelo usuário em seu primeiro acesso.
RS11	Permitir que usuários alterem suas próprias senhas, sem precisar da intervenção do administrador.
RS12	Impedir acesso simultâneo ao sistema utilizando uma mesma conta em máquinas diferentes.
RS13	Validar o campo CPF.
RS14	Impedir cadastro de dois participantes com CPF iguais.
RS15	Excluir participantes que não confirmaram a inscrição em 5 dias úteis após o cadastro.
RS16	O sistema deve ficar disponível por pelo menos 9 horas por dia.
RS17	Garantir que o sistema suporte vários acessos simultâneos.
RS18	Impedir SQL Injection.

Fonte: Própria

4.3 CASOS DE TESTES

Baseando-se nos requisitos de segurança, foram elaborados casos de testes que definem as prováveis entradas para cada teste e aquilo que se espera como saída. Contudo, não é possível elaborar um teste para todos os requisitos, pois alguns deles não possuem uma entrada que pode ser definida ou explicada, como, por exemplo, utilizar o protocolo HTTPS, para testar isso bastar observar a URL do sistema. Deste modo, nem todos os requisitos possuem um caso de teste, ficando no final do capítulo um relatório explicando se eles foram cumpridos ou não.

Tabela 6: Casos de testes

Área do sistema	Requisito	Cód	Teste	Entrada	Saída Esperada
Admin.	RS1	T1	Efetuar login	Usuário e senha válidos	Conseguir acesso à página
				Usuário e senha inválidos	Impedir acesso e exibir mensagem de erro de autenticação.
				Usuário válido e senha inválida	
				Usuário inválido e senha válida	
	RS3	T2	Criar usuário	Nome de usuário qualquer e senha contendo caracteres, números e caracteres especiais	Usuário criado com sucesso
				Nome de usuário qualquer e senha contendo apenas 3 caracteres simples	Exibir mensagem pedindo para inserir uma senha maior
				Nome de usuário qualquer e senha contendo apenas números ou apenas caracteres	Exibir mensagem dizendo que esta senha é muito simples dando a opção de continuar mesmo assim.
				Nome de usuário qualquer e senha vazia	Exibir mensagem pedindo para preencher o campo senha
				Nome de usuário vazio e senha complexa	Exibir mensagem pedindo para preencher o campo usuário
				Nome de usuário com mais de 30 caracteres	Exibir mensagem dizendo que o nome de usuário é muito longo
	RS2	T3	Permissão para criar usuários	Utilizando conta do Gelato, escolher opção criar usuário	Impedir o acesso e mostrar mensagem de erro
				Utilizando conta de admin. sem	Impedir a ação e exibir mensagem

				permissão de criação, escolher opção criar usuário	de erro
				Utilizando conta de admin. com permissão de criação, escolher opção criar usuário	Permitir a ação e levar até a tela de criação.
				Utilizando conta de admin. com permissão, escolher criar um usuário *root (*superusuário)	Impedir a ação e exibir mensagem de erro
	RS2	T4	Permissão para excluir usuários	Utilizando conta do Gelato, escolher opção excluir usuário	Impedir o acesso e exibir mensagem de erro
				Utilizando conta de admin. sem permissão de exclusão, escolher opção excluir usuário	Impedir a ação e exibir mensagem de erro
				Utilizando conta de admin. com permissão de exclusão, escolher opção excluir usuário	Permitir a ação
				Utilizando conta de admin. com permissão de exclusão, escolher opção excluir o usuário *root* (superusuário)	Impedir a ação
	RS2/ RS11	T5	Permissão para editar usuários	Utilizando conta do gelato, escolher opção para editar um usuário qualquer	Impedir o acesso e exibir mensagem de erro
				Utilizando conta do Gelato, escolher opção para editar o próprio usuário	Permitir a alteração de senha e dados pessoais
				Utilizando conta de admin. sem permissão, escolher	Impedir a ação e mostrar mensagem de erro

				opção editar usuário	
				Utilizando conta de admin. com permissão, escolher opção para editar usuário qualquer	Permitir ação
				Utilizando conta de admin. com permissão, escolher opção editar o usuário *root* (superusuário)	Impedir a ação
	RS2	T6	Permissão para realizar uma ação em um evento	Utilizando conta de admin. com permissão, realizar uma ação em eventos	Permitir a ação
				Utilizando conta de admin. sem permissão, realizar uma ação em eventos	Impedir a ação e exibir mensagem de erro
				Utilizando conta do Gelato, realizar uma ação em eventos	Impedir o acesso e mostrar mensagem de erro
	RS2	T7	Permissão para realizar uma ação em um participante ou inscrito	Utilizando conta do Gelato, realizar uma ação em participantes e em inscritos	Impedir o acesso e exibir mensagem de erro
				Utilizando conta de admin. sem permissão, realizar uma ação em participantes e em inscritos	Impedir ação e mostrar mensagem de erro
				Utilizando conta de admin. com permissão, realizar uma ação em participantes e em inscritos	Permitir a ação
Gelato	RS1	T8	Efetuar login	Usuário e senha válidos	Conseguir acesso à página

				Usuário e senha inválidos	Impedir acesso e exibir mensagem de erro de autenticação.
				Usuário válido e senha inválida	
				Usuário inválido e senha válida	
Cadastro	RS13/ RS14	T9	Validar CPF	Inserir um CPF inválido	Exibir mensagem de erro
				Inserir um CPF válido	Aceitar o CPF
				Inserir CPF já cadastrado	Exibir mensagem de erro
Todos	RS18	T10	SQL Injection	Inserir 'OR '1' = '1 em um campo senha	Impedir o acesso e exibir mensagem de erro
				Inserir teste';-- em um campo usuário	Impedir o acesso e exibir mensagem de erro
				Inserir asd';drop table users;	Impedir a ação e mostrar mensagem de erro

Fonte: Própria

4.4 RESULTADOS OBTIDOS

Após executar todos os testes previstos no caso de testes, todos os resultados obtidos foram observados, desta forma, é possível compará-los com aquilo que era esperado.

Começando pelo teste com código T1, todas as entradas foram testadas e o resultado obtido condiz com o esperado. Ao deixar de preencher tanto o campo 'Usuário' como o campo 'Senha' é exibido uma mensagem informando a sua obrigatoriedade. Já quando um dos campos é preenchido com dados inválidos, a mensagem mostrada solicita um usuário ou uma senha correta, informando ainda a sensibilidade à maiúsculas e minúsculas.

No teste T2, os resultados não foram tão satisfatórios como no anterior. Aqui

todas as entradas foram testadas, porém nem todas as saídas obtidas corresponderam com o esperado. O fato de deixar tanto o campo 'Usuário' quanto o campo 'Senha' vazios condiz com o planejado, em que é exibido uma mensagem dizendo que estes campos devem ser preenchidos. Por outro lado, a complexidade da senha não é levado em conta, assim o sistema permite a criação de usuários com a senha “s”, sem demonstrar nenhuma objeção quanto a isso. Já quanto ao limite de caracteres, o sistema possui uma máscara no campo 'Usuário' impedindo que este ultrapasse 30 caracteres.

Em relação ao teste T3, referente à permissões para criar contas de usuários, os resultados obtidos foram quase perfeitos. Ao logar-se com o usuário sem permissão, não é possível executar a ação de criar uma conta, é exibido uma mensagem dizendo “Você não tem permissão para edição”. Agora, quando se utiliza um usuário com permissão, a criação é feita com sucesso. Já em relação ao usuário do Gelato, este não consegue nem entrar na administração. O único fator que não foi cumprido é a possibilidade de criar um superusuário (*root*[3]) utilizando um usuário de administrador comum que apenas possui permissão de criação.

No teste T4, semelhante ao teste anterior, apenas o usuário com permissão consegue excluir uma conta existente, o usuário do Gelato nem consegue acessar a administração. Não obstante, este usuário pode excluir o superusuário (*root*) sem se preocupar com nada.

O teste T5 obteve resultados iguais ao teste T4, em que o usuário do Gelato não possui acesso a administração, impedindo que este altere seus próprios dados pessoais ou sua senha, e apenas aquele que possui permissões pode editar outros usuários, inclusive o *root* (superusuário).

Tanto o teste T6 como o T7 possuíram resultados que satisfizeram o esperado, em que, apenas usuários administradores que possuem permissão conseguiram executar qualquer tipo de ação (criar, excluir ou editar) nos eventos e

3 O superusuário *root* possui permissão para fazer tudo no sistema, ele é o primeiro usuário criado e, geralmente, pertence ao líder da equipe de desenvolvimento.

nos participantes, e, o usuário do Gelato não obteve acesso ao Administrativo.

O T8 também obteve todos os resultados esperados, impossibilitando acesso com usuário e/ou senha inválidos, impedindo campos vazios, e permitindo acesso quando tudo preenchido de forma correta. Neste caso, as contas de administradores também possuem acesso ao Gelato.

O teste T9 diz respeito ao cadastro que é aberto ao público em geral. Nesta parte do teste é validado se o campo CPF, que é único para cada pessoa, é preenchido de forma correta, ou seja, se é um CPF válido, e se não houve mais de um cadastro utilizando o mesmo número. Todos os resultados obtidos foram satisfatórios, pois o campo possui uma máscara que verifica a validade do CPF digitado e ainda compara se não houve um cadastro anterior utilizando o mesmo documento. Assim sendo, não há uma perda de integridade nas informações das pessoas que se inscreveram no evento.

Por último é efetuado o teste T10 que busca verificar a possibilidade de executar um *SQL Injection* no sistema. Aqui foram efetuadas entradas mais conhecidas que podem ser encontradas em qualquer fórum ou site informativo, e, de forma satisfatória, os resultados estão de acordo com o planejado. Visto que, o Django, *framework* utilizado no desenvolvimento do sistema, foi criado visando minimizar as vulnerabilidades mais comuns em sistemas *web*, não há meios de executar um *SQL Injection* que já seja de conhecimento mútuo.

4.5 TESTE ESTRUTURAL

O *framework* Django foi desenvolvido para se programar em camadas que são divididas em três, *model, views e template*. Seu objetivo é agilizar o trabalho dos desenvolvedores e ajudá-los a cumprir *deadlines,* além de possuir grande foco na segurança, evitando erros comuns de segurança, como, por exemplo, *SQL Injections.*

O sistema *web* em estudo utiliza-se deste *framework*, deste modo, sua codificação possui todas as três camadas citadas anteriormente. O *model layer* trabalha com as classes, é aqui onde são criadas as classes utilizadas no sistema, sendo elas 'Participantes', 'Evento' e 'Inscricao'. Já a *views layer* é onde são desenvolvidas as funções de cadastros, login, envio de e-mail, consultas, toda a parte funcional do sistema. Por último, o *template layer* é a camada que implementar os html's que são requisitadas pela *views*.

Tendo em vista o Gelato que necessita de autenticação para realizar suas ações, pode-se observar nas funções como, por exemplo 'lista_inscritos', visto logo abaixo, uma verificação logo antes de realizar qualquer coisa:

Figura 1: Função lista_inscritos

```
def lista_inscritos(request):
    if request.user.is_authenticated():
        c = Collator()
        inscritos = sorted(Inscricao.objects.all(),key=lambda x: (c.sort_key(x.participante.nome_completo)))
        total_inscritos = len(Inscricao.objects.all())
        template = loader.get_template('gelato/lista_inscritos.html')
        context = RequestContext(request,{'inscritos' : inscritos,'total_inscritos':total_inscritos})
        return HttpResponse(template.render(context))
    return redirect("/gelato/logme/")
```

Fonte: Equipe de desenvolvimento Escola de Primavera

No código acima existe uma condição que verifica se o usuário está autenticado, e, caso não esteja, é retornado um redirecionamento para outra função, o 'logme', desta forma ele evita que aqueles não logados no sistema acessem informações de inscritos no evento. Agora, quando o usuário está autenticado é renderizado o template 'lista_inscritos.html'.

A função 'logme' serve para verificar se o usuário está autenticado e, caso verdadeiro, redirecionado para a página, e, caso falso, redirecionado para o *login.* Se houver algum erro, é executado uma exceção.

Figura 2: Função logme

```
def logme(request):
    if request.user.is_authenticated():
        return redirect("/gelato/main/")
    template = loader.get_template('gelato/logme.html')
    erro = request.session.get('erro_login')
    context = RequestContext(request,{'erro' : erro})
    return HttpResponse(template.render(context))
```

Fonte: Equipe de desenvolvimento Escola de Primavera

A função 'log_in' irá verificar aquilo que foi preenchido nos campos 'Usuário' e 'Senha' do formulário e verificar se estes condizem com o armazenado no banco de dados, gerando uma exceção caso ocorra algum erro de autenticação, e redirecionando para a função 'main' se a autenticação for um sucesso.

Figura 3: Função log_in

```
def log_in(request):
    username = request.POST['username']
    password = request.POST['password']
    user = authenticate(username=username, password=password)
    try:
        login(request, user)
        request.session['erro_login'] = False
    except:
        request.session['erro_login'] = True
        pass
    return redirect("/gelato/main")
```

Fonte: Equipe de desenvolvimento Escola de Primavera

A função 'main' também realiza uma verificação logo no começo para ver se o usuário está realmente autenticado, redirecionando para o 'logme' se não estiver. Confirmando a autenticação, a função irá carregar todos os objetos de 'eventos' e irá renderizar o template 'main.html'.

Figura 4: Função main

```
def main(request):
    if request.user.is_authenticated():
        total_inscritos = len(Inscricao.objects.all())
        eventos = Evento.objects.all()
        template = loader.get_template('gelato/main.html')
        context = RequestContext(request,{'total_inscritos' : total_inscritos,'eventos' : eventos})
        return HttpResponse(template.render(context))
    return redirect("/gelato/logme/")
```

Fonte: Equipe de desenvolvimento Escola de Primavera

4.6 AVALIAÇÃO DOS REQUISITOS

Após a realização dos testes sobre os requisitos de segurança planejados no início do capítulo, alguns deles previstos no caso de testes, nota-se que o sistema em estudo cumpre grande parte deles, porém deixa a desejar em relação a alguns.

O requisito RS1, que diz respeito a autenticação de usuário e senha, é cumprido com êxito no sistema como pode ser visto nos testes T1 e T8, em que o acesso ao Gelato e a administração só é permitida caso a conta seja válida, lembrando que usuário do Gelato não podem acessar a administração, mas o oposto é possível.

Quanto ao requisito RS2, o sistema cumpre de forma parcial o planejado, pois, mesmo criando permissões e atribuindo elas para usuários específicos, o superusuário (*root*) que detem um poder maior, ainda pode ser excluído e editado por qualquer outro administrador que possua permissão, sendo que, o ideal, é o superusuário possuir uma autoridade maior que os demais administradores, impossibilitando qualquer ação sobre ele. Tirando este fato, todos os outros fatores são respeitados, onde apenas administradores com permissão possuem permissão para executar ações na administração e os usuários do Gelato não têm acesso a esta parte.

O sistema não cumpre totalmente com o requisito RS3, pois ao criar uma conta, mesmo evitando deixar campos de 'Usuário' e 'Senha' vazios, não se queixa

quando uma senha é simples demais, ou seja, a senha pode conter apenas um carácter e a conta é criada com sucesso.

O requisito RS4 não é desempenhado com sucesso, uma vez que ao se criar uma conta, embora esta nunca seja utilizada, ela ficará no sistema até ser excluída ou classificada como inativa de forma manual por outro usuário.

O requisito RS5 também não é exercida, dado que uma sessão nunca é expirada, mesmo que fique inativa por mais de um dia. Para que seja solicitado novamente o usuário e a senha é necessário sair manualmente do sistema.

Tanto o requisito RS6 quanto o RS7 são muito bem desempenhados pelo sistema. Todos os campos de senhas possuem uma mascará impedindo de serem visualizada por outras pessoas que se encontram juntas com o usuário, e o desenvolvedor levou em conta a segurança das informações que são trocadas entre o cliente e o servidor ao implementar o protocolo HTTPS no sistema.

O requisito RS8 diz respeito a *logs* gerados pelo sistema para facilitar a sua auditoria. Observou-se durante todo o estudo de caso, quando eram executadas diversas ações no sistema, que são gravadas todas as ações do usuário e estas listadas em sua página inicial, assim sendo, este requisito é desempenhado com sucesso.

Quanto ao requisito RS9, como o Django divide toda a codificação do *software* em diversas camadas e arquivos, senhas necessárias pelo sistema, como o do banco de dados, são armazenadas em um único arquivo de configuração que não pode ser acessado facilmente. Sendo assim, pode-se dizer que este requisito é cumprido pelo sistema.

Os requisitos RS10 e RS11 não são exercidas de modo satisfatório, uma vez que o único modo de executar uma alteração em qualquer conta, até mesmo a própria, é possuindo permissão para tal, Deste modo, usuário do Gelato ou

administradores sem permissão de edição, não podem alterar suas próprias senhas, necessitando contato com algum indivíduo que possua tal permissão.

O requisito RS12 não é atendido pelo sistema, pois é possível, utilizando o mesmo usuário, logar-se várias vezes ao mesmo tempo e de lugares diferentes, ou seja, é permitido acessar o sistema com o mesmo usuário pelo computador pessoal, pelo *tablet* e pelo *smartphone* simultaneamente.

Os requisitos RS13 e RS14 são cumpridos pelo sistema como pode ser visto no teste T9, em que é impedido o cadastro de dois CPF iguais e a inserção de dados inválidos, dado que o formulário possui uma máscara impedindo o preenchimento indevido do campo.

O sistema não exerce exatamente o que é definido no requisito RS15 que é excluir participantes que não confirmaram a inscrição em 5 dias úteis, pois ele envia um e-mail lembrando estas pessoas que é necessário realizar uma confirmação no *site*. Esta forma de abordagem é muito melhor comparada com o planejado, visto que a remoção só acontecerá se o participante afirmar sua desistência ao administrado.

Quanto os requisitos RS16 e RS17, referente a disponibilidade das informações, durante todo o processo de estudo de caso não houve nenhuma queda do sistema e, levando em conta que o evento já passou e houve diversos inscritos, pode-se dizer que o sistema aguenta uma boa quantidade de acessos ao mesmo tempo.

Por último, em relação ao requisito RS18, já foi mencionado anteriormente que o *framework* Django foi criado para evitar as fraquezas mais conhecidas e uma delas é o *SQL Injection,* deste modo, é muito difícil executar este tipo de ação, a menos que seja algo novo e inovador, nunca descoberto antes.

Como forma de finalizar esta análise é necessário observar que este *software*

cumpre de forma satisfatória onze dos requisitos planejados e faz uso de uma tecnologia com foco na segurança. Não obstante, ele apresenta diversas fraquezas em relação a confidencialidade da informação ao permitir que uma sessão dure diversos dias ou que possa ser efetuado acessos simultâneos com a mesma conta. Sendo necessário exercer algumas melhorias no sistema para minimizar tais vulnerabilidades.

5 CONSIDERAÇÕES FINAIS

Nos dias atuais, a informação é um bem de valor inestimável e, muitas vezes, confidenciais para as organizações e para os indivíduos. Devido a isso, a segurança passou a ser indispensável para o desenvolvimento de um *software,* necessitando-se utilizar de todos os meios possíveis para garanti-la em perfeitas condições, sendo os testes um destes meios.

Os testes trazem diversos benefícios para a segurança, pois estes possuem como objetivo encontrar fraquezas. Deste modo, ao elaborar um ótimo processo de testes pode-se localizar diversas vulnerabilidades na segurança de um sistema ou programa que seriam muito prejudiciais caso passassem despercebidas para o usuário final, permitindo que indivíduos fizessem uso deles para fins maliciosos.

Entretanto, um *software* não é considerado seguro apenas porque foram executados diversos testes nele. Vale ressaltar que o objetivo é encontrar e não solucionar, cabendo aos desenvolvedores otimizarem aquilo que criaram com a intenção de consertar aquilo que foi detectado. Sendo assim, a importância dos testes para a segurança é o fato destes facilitarem a ação de minimizar vulnerabilidades existentes.

Uma forma de exemplificar o benefício citado acima é o seguinte fato, o desenvolvedor do sistema utilizado como estudo de caso aprovou aquilo que foi realizado aqui e solicitou um relatório de tudo para que, deste modo, ele possa otimizar seus projetos futuros.

Não obstante, planejar os requisitos de segurança e os casos de testes requer muito tempo e esforço e isto, de um modo ou de outro, acabou afetando o estudo de caso realizado, pois este poderia ser mais profundo e complexo. Caso houvesse um maior período para elaboração deste trabalho, com certeza muitas

coisas seriam exploradas, como o uso de ferramentas de testes, a elaboração de mais requisitos focados no servidor e a criação de um caso de testes que forçaria ao limite o sistema estudado.

6 REFERÊNCIAS

COMMON WEAKNESS ENUMERATION. **2011 CWE/SANS top 25 most dangerous software errors**. 2011. Disponível em <https://cwe.mitre.org/top25/archive/2011/2011_cwe_sans_top25.pdf>. Acesso em: 21 out. 2015, às 20h29min

DELAMARO, Márcio Eduardo; MALDONADO, José Carlos; JINO, Mario. **Introdução ao teste de software**. Rio de Janeiro : Elsevier, 2007

FERREIRA, Aurélio Buarque de Holanda. **Dicionário Aurélio da Língua Portuguesa**. 5ª ed. Curitiba : Positivo, 2010, p. 1906.

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION / INTERNATIONAL ELECTROTECHNICAL COMMISSION, **ISO/IEC 15408-1:2009**: Information technology : Security techniques : Evaluation criteria for IT security : Part 1 : Introduction and general model. 2009. Disponível em: <http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=50341>. Acesso em: 15 out. 2015, às 17h39min.

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION / INTERNATIONAL ELECTROTECHNICAL COMMISSION, **ISO/IEC 15408-2:2008**: Information technology : Security techniques : Evaluation criteria for IT security : Part 2 : Security functional components. 2008. Disponível em: <http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=46414>. Acesso em: 15 out. 2015, às 17h41min.

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION / INTERNATIONAL ELECTROTECHNICAL COMMISSION, **ISO/IEC 15408-3:2008**: Information technology : Security techniques : Evaluation criteria for IT security : Part 3 : Security assurance components. 2008. Disponível em: <http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=46413>. Acesso em: 15 out. 2015, às 17h43min.

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION / INTERNATIONAL ELECTROTECHNICAL COMMISSION, **ISO/IEC 27001:2013**: Information technology : Security techniques : Information security management systems. 2013. Disponível em: <http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=54534>. Acesso em: 15 out. 2015, às 17h32min.

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION / INTERNATIONAL ELECTROTECHNICAL COMMISSION, **ISO/IEC 27002:2013:** Information technology : Security techniques : Code of practice for information security controls. 2013. Disponível em: <http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=54533>. Acesso em: 15 out. 2015, às 17h35min.

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION / INTERNATIONAL ELECTROTECHNICAL COMMISSION, **ISO/IEC 27005:2011:** Information technology : Security techniques : Information security risk management. 2011. Disponível em: <http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=56742>. Acesso em: 15 out. 2015, às 17h36min.

KIM, David; SOLOMON, Michael G. **Fundamentos de segurança de sistemas de informação**. Trad. Daniel Vieira. 1ª ed. Rio de Janeiro : LTC, 2014.

LYRA, Maurício Rocha. **Segurança e auditoria em sistemas de informação**. Rio de Janeiro : Editora Ciência Moderna Ltda., 2008.

RIOS, Emerson; MOREIRA, Trayahú. **Teste de software**. 2ª ed. rev. ap. Rio de Janeiro : Editora Alta Books, 2006.

SILVA, José Gonçalo. **Testes de segurança :** Conferência Código Seguro. 2004. Disponível em: <http://www.dei.estg.ipleiria.pt/eventos/conferencias/csi2004/apresentacoes/testessegurança.pdf>. Acesso em: 20 out. 2015, às 19h43min.