UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE ENGENHARIA DE COMPUTAÇÃO

PABLO CAIÃ DE MELLO SOARES

O Processo de Testes de Intrusão no CPD-UFRGS

Monografia apresentada como requisito parcial para a obtenção do grau de Bacharel em Engenharia da Computação

Orientador: Prof. Dr. Raul Weber

Porto Alegre
2017

AGRADECIMENTOS

Agradeço primeiramente aos meu pais, José V. F. Soares e Loiva I. de M. Soares, por terem me apoiado e incentivado incondicionalmente, sem vocês não teria sido possível a conclusão desta etapa. Ao meu irmão, Leonardo, e a minha irmã, Aline, que deixou muitas saudades, por todo companheirismo, além da minha sobrinha Tainá, por toda sua alegria contagiante. Vocês são minha família e motivo de muito orgulho.

Devo agradecer a minha namorada, Tyana Oliveira, por sempre estar ao meu lado. Seu carinho, suporte e compreensão, além de toda dedicação por seus objetivos, me motivam a ser uma pessoa melhor.

Agradeço ao Engº André Machado, amigo da família, por se dispor a lecionar aulas de matemática e física voluntariamente. Seus ensinamentos e experiência possibilitaram superar dificuldades de aprendizado e me incentivaram a cursar uma engenharia.

Ao meu orientador, Raul Weber, agradeço pelas discussões sobre segurança e sugestões que auxiliaram no desenvolvimento deste trabalho. Agradeço também aos colegas do Departamento de Segurança da Informação do CPD, o aprendizado absorvido devido aos seus conhecimentos e pelos desafios enfrentados pela equipe permitiram me desenvolver tecnicamente, o que possibilitou a concepção deste trabalho.

Por fim, agradeço aos meus amigos que, de uma forma ou de outra, fizeram parte da minha formação pessoal, profissional e acadêmica. Em especial, gostaria de agradecer: William Gomes e William Vidal, por serem grandes amigos que conheci no curso, e também ao Fabio Cavalheiro, Rafael Gomes e Wyllian Hossein.

RESUMO

Vulnerabilidades presentes em aplicações web são amplamente utilizadas por atacantes para o roubo de informações e invasões a redes de computadores. O fato de estarem expostas a milhões de usuários pela Internet evidencia que segurança é um requisito indispensável no projeto de tais aplicações.

A partir de testes de intrusão, é possível descobrir vulnerabilidades e, posteriormente, corrigi-las, constituindo-se de uma ação preventiva a invasões. A proposta deste trabalho é ilustrar como essa técnica é aplicada aos sites da UFRGS pela equipe de segurança do CPD, justificando sua escolha e necessidade de adaptação, formalizando um processo para essa atividade, e analisando sua efetividade através de avaliações experimentais.

Palavras-chave: Pentest. Aplicações Web. Segurança da Informação. Vulnerabilidades de Segurança. Redes de Computadores.

The Intrusion Testing Process of CPD-UFRGS

ABSTRACT

Vulnerabilities present in web applications are widely used by attackers for information theft and hacking into computer networks The fact that they are exposed to millions of users through the Internet shows that security is an indispensable requirement in the design of such applications.

From intrusion tests, It is possible to discover vulnerabilities and subsequently correcting them, constituting a preventive action against invasions. The proposal of this work is to illustrate how this technique is applied to the UFRGS sites by the CPD security team, justifying It's choice and need for adaptation, formalizing a process for this activity and analyzing its effectiveness through experimental evaluations.

Keywords: Pentest, Web Applications, Information Security, Security Vulnerabilities, Computer Networks.

LISTA DE ABREVIATURAS E SIGLAS

AJAX	Asynchronous Javascript and XML
CMS	Content Management System
CPD	Centro de Processamento de Dados
CSS	Cascading Style Sheets
CSRF	Cross-Site Request Forgery
CVE	Common Vulnerabilities and Exposires
CVSS	Common Vulnerability Scoring System
DNS	Domain Name System
DSInf	Departamento de Segurança da Informação
DOM	Document Object Model
DVWA	Dawn Vulnerable Web Application
HTTP	HyperText Transfer Protocol
HTTPS	HyperText Transfer Protocol Secure
HTML	HyperText Markup Language
IDS	Intrusion Detection System
IPS	Intrusion Prevention System
JSON	JavaScript Object Notation
LFI	Local File Inclusion
NoSQL	Not Only Structured Query Language
OWASP	Open Web Application Security Project
PSI	Política de Segurança da Informação
REST	Representational State Transfer
RFI	Remote File Inclusion
RFC	Request For Comments

SGBD	Sistema de Gerenciamento de Banco de Dados
SMTP	Simple Mail Transfer Protocol
SQL	Structured Query Language
SSL	Secure Socket Layer
TCP	Transmission Control Protocol
TI	Tecnologia da Informação
TRI	Time de Resposta a Incidentes
UDP	User Datagram Protocol
URL	Uniform Resource Locator
WAF	Web Application Firewall
XSS	Cross-Site Scripting
ZAP	Zed Attack Proxy

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Teste de intrusão, também chamado de teste de invasão, teste de penetração ou pentest, é um método utilizado para verificar a segurança de um ambiente, plataforma ou sistema, por meio da simulação de ataques reais explorando as vulnerabilidades encontradas (UTO, 2013). Quando o ambiente para testes se restringe a aplicações web, esse método deve ser efetuado de forma mais específica, considerando as características inerentes a esses sistemas (MEUCCI; MULLER, 2015).

O fato de tais aplicações estarem conectadas à Internet as tornam inseguras por padrão, pois está sendo permitido o acesso direto de desconhecidos ao servidor hospedeiro (BALLAD; BALLAD, 2009). Caso exista um *firewall*, por exemplo, exceções precisam ser criadas para que acessos externos sejam permitidos nesse servidor, que pode estar conectado a outros servidores da mesma rede, como servidores de banco de dados. Esses servidores, normalmente não são acessíveis pela Internet por medidas de segurança com *firewalls* (NIC-BR, 2003). Ou seja, a segurança de todo ambiente é prejudicada para tornar a aplicação disponível. Portanto, se uma aplicação for insegura, atacantes podem ganhar acesso e controle de servidores de uma rede através de suas vulnerabilidades, burlando medidas de segurança física e de rede (BALLAD; BALLAD, 2009).

Para validar a segurança de uma aplicação web, e se prevenir de ataques, cujas motivações mais comuns são financeiras, de prestígio, de demonstração de poder, ideológicas ou comerciais (CERT-BR, 2012), o guia de testes da OWASP[1] (MEUCCI; MULLER, 2015) estabelece um arcabouço que abrange e defende a necessidade de diversos tipos de testes de segurança diferentes conforme a fase do ciclo de vida do desenvolvimento de um software. O mesmo guia sugere que testes de intrusão sejam realizados na fase de implantação[2].

No âmbito da UFRGS, para testar a segurança de sites desenvolvidos pelas diversas unidades da Universidade, o Departamento de Segurança da Informação (DSInf) do Centro de Processamento de Dados (CPD), unidade responsável por prover serviços de Tecnologia da Informação (TI), utiliza de testes de intrusão para realizar tal tarefa. A método de testes de segurança utilizada se resume aos testes de intrusão pelo fato de não existir, por diversos fatores, um acompanhamento dos projetos de aplicações web desde sua concepção. Outra justificativa para a escolha é o fato de um número considerável de

[1] The Open Web Application Security Project: https://www.owasp.org/

[2] Fase do ciclo de vida de um software, no contexto de um Sistema de Informação, que corresponde textualmente à passagem do software para a produção.

sites estarem hospedados nas próprias unidades, o que dificulta a obtenção de códigos-fonte, impossibilitando o uso de técnicas como revisão de código, por exemplo.

A necessidade de haver auditoria de segurança se justifica pelo fato de que não existir um acompanhamento por parte do CPD em relação a aplicação desse requisito no projeto e desenvolvimento desses sistemas, feito por bolsistas, funcionários ou empresas terceirizadas, em que não se há garantia sobre suas noções de desenvolvimento seguro. Infelizmente, tal problema entra de acordo com WIESMANN et al. (2005), que afirmam não existir, ainda, na maioria das empresas e projetistas de software, uma movimentação forte em prol da segurança de TI.

A responsabilidade que o DSInf possui para auditar esses sites e averiguar se estão em conformidade com a Política de Segurança da Informação (PSI) da UFRGS (CONSUN, 2014) é garantida pelo item III do Art. 5º da PSI, que define a seguinte atividade: *"monitorar, auditar e avaliar periodicamente as práticas de segurança da informação adotadas pela Universidade"*. Nesse documento, ainda é garantido, através de um parágrafo único, o direito de solicitar por medidas para garantir que essas práticas sejam seguidas: *"Cabe às demais unidades da Universidade, no âmbito de suas competências, a implementação e o acompanhamento de ações para segurança da informação"*.

Levando em conta a grande quantidade de sites da Universidade que precisam ser testados, e o fato do DSInf possuir diversas demandas e poucos membros, tal tarefa é reservada a bolsistas, cuja a rotatividade é muito maior que a de funcionários. Essa rotatividade impede que procedimentos muito complexos sejam repassados a bolsistas, devido ao tempo de aprendizado necessário. Uma estratégia de testes, que leva em consideração os problemas apresentados, é apresentada e avaliada neste trabalho.

1.1 Objetivos do trabalho

Tendo em vista a quantidade de aprendizado necessário para se dominar a técnica de testes de intrusão, este trabalho visa a criação de uma estratégia de testes de intrusão direcionada ao ambiente da UFRGS, visando um aprendizado agilizado, e por partes, sendo aplicável mesmo sem seu total domínio. Para isso, a literatura da área será estudada e as metodologias, ou estratégias, de teste existentes, que apesar de serem mais completas, necessitam de mais tempo de aprendizado, serão aplicadas aos sites da universidade. Dessa forma, será possível visualizar otimizações para uma estratégia direcionada devido à inexistência de certas vulnerabilidades, ou por características comuns entre as diversas

aplicações web da UFRGS.

Além disso, a eficiência dessa estratégia será avaliada através de seu uso por bolsistas do DSInf. Essa avaliação permitirá analisar de forma quantitativa a facilidade de aprendizagem e aplicação da solução comparando bolsistas com diferentes níveis de experiência em aplicações de diferentes níveis de complexidade.

1.2 Organização do texto

Os primeiros capítulos apresentam fundamentos e justificativas para a realização deste trabalho. De maneira mais detalhada, o capítulo 2 introduz todos os conceitos necessários sobre técnicas de teste de segurança (sendo aprofundando o teste de intrusão) e aplicações web, fundamentais para o entendimento do trabalho. No capítulo 3, é discutido a necessidade de uma estratégia de testes de intrusão customizada, e documentado todo o processo de desenvolvimento.

A partir do capítulo 4, o resultado desse processo é apresentado, para posterior avaliação. O capítulo 5 apresenta essa avaliação, de caráter experimental, e a discussão dos resultados. Por fim, no capítulo 6, são ressaltados os principais desafios e contribuições deste trabalho, assim como trabalhos futuros.

2 FUNDAMENTAÇÃO E CONCEITUAÇÃO

Para haver segurança real em uma aplicação web, Ballad e Ballad (2009) afirmam, ironicamente, que a solução seria não expô-la a Internet. Essa solução é impraticável, visto que o objetivo desses sistemas é prover serviços para serem acessados remotamente. Tais serviços compreendem desde a interfaces de configuração de equipamentos de rede a *blogs*, *Internet Banking*, entre tantos outros. Em termos de engenharia de software, conforme Sommerville (2003, p. 85), os requisitos não funcionais são aqueles que não dizem respeito diretamente às funções específicas fornecidas pelo sistema. Segurança é um exemplo de requisito não funcional considerado fundamental em aplicações de comércio eletrônico e outras similares no âmbito da Internet (FIORIO et al., 2007).

Como forma de verificar a segurança de aplicações web, existem diversas técnicas de teste disponíveis para identificar vulnerabilidades. Neste capítulo, serão apresentadas essas técnicas, sendo aprofundado o teste de intrusão, que posteriormente será justificado sua escolha. Também será introduzido os principais conceitos relacionados à aplicações web atuais, que são os ativos[1] de interesse deste trabalho. Os fundamentos deste capítulo são essenciais para o entendimento da proposta do trabalho.

2.1 Técnicas de teste de segurança

O objetivo desta seção, que tem como base o guia de testes da OWASP - *"OWASP Testing Guide 4.0"* (MEUCCI; MULLER, 2015), é apresentar as principais técnicas existentes de teste de segurança em aplicações, para posteriormente poder entender, de forma mais detalhada, a escolha do teste de intrusão como forma de verificar a segurança dos sites da Universidade. Segundo os autores desse guia, cada uma dessas técnicas possuem suas características como conhecimento técnico necessário para efetuar o teste, tempo de execução, a partir de que fase do ciclo de desenvolvimento de software pode ser aplicado, além das suas vantagens e desvantagens.

[1] Um ativo de TI é qualquer informação, sistema ou hardware de propriedade de uma empresa que é utilizado no decurso das atividades empresariais.

2.1.1 Inspeção e revisão manual

Inspeções manuais são avaliações que normalmente testam as implicações de pessoas, políticas e processos na segurança da informação (MEUCCI; MULLER, 2015). Essas avaliações são geralmente conduzidas por análise de documentação, inspeção de decisões de tecnologia e *design* da arquitetura do sistema ou na realização de entrevistas com desenvolvedores ou proprietários do sistema.

Essas inspeções são particularmente eficientes para testar se as pessoas envolvidas em um projeto de software compreendem o processo de segurança, se foram informados da política de segurança da informação, e se possuem as habilidades necessárias para projetar e implementar uma aplicação segura (MEUCCI; MULLER, 2015). Outras atividades, incluindo a revisão da documentação, das políticas de desenvolvimento seguro, requisitos de segurança e *design* da arquitetura da aplicação, devem todos ser realizados utilizando inspeções manuais.

2.1.2 Modelagem de riscos estruturada a ameaças

A modelagem de ameaças é uma técnica utilizada para se identificar ameaças de segurança que sistemas e aplicações podem vir a enfrentar. Para desenvolver um modelo de ameaça, existe a abordagem proposta pelo padrão NIST 800-30 (NIST, 2012) para avaliação de risco. Essa abordagem envolve ações como:

- Decompor a aplicação para entender seu funcionamento, seus ativos e conectividade;
- Definir e classificar ativos, de acordo com a importância do ativo para o negócio e tangibilidade;
- Explorar potenciais vulnerabilidades, sejam elas técnicas, operacionais ou de gerenciamento;
- Explorar potenciais ameaças planejando possíveis vetores de ataques vistos da perspectiva do atacante;
- Criar estratégias de mitigação, desenvolver controles de mitigação para cada ameaça que se considerar real.

Essa técnica geralmente é utilizada por projetistas de sistemas com conhecimentos em segurança e pode ser vista como uma avaliação de riscos para aplicações (MEUCCI;

MULLER, 2015). Modelagem de ameaças permite que se possa desenvolver estratégias de mitigação de potenciais vulnerabilidades e pode ser aplicada desde o início do projeto de uma aplicação.

2.1.3 Revisão de código

A revisão de código fonte é o processo em que se verifica o código de uma aplicação por motivações de segurança. Essa técnica é considerada a mais efetiva para se detectar vulnerabilidades nas fases iniciais do desenvolvimento de uma aplicação (CONKLIN; ROBINSON, 2016), além de muitas vulnerabilidades graves não poderem ser detectadas com qualquer outra forma de teste (MEUCCI; MULLER, 2015). O guia de testes da OWASP cita o fato da maioria dos especialistas em segurança concordarem que não há um método equivalente que substitui a revisão de código pois todas as informações para a identificação de problemas de segurança estão em algum lugar no código.

Diversos problemas de segurança são mais difíceis de se descobrir com outras técnicas, tais como teste de intrusão (CONKLIN; ROBINSON, 2016). Entretanto, tal técnica costuma ser mais demorada (BIRD, 2016), o que pode tornar proibitivo sua aplicação em sistemas mais complexos, devido a grande quantidade de tempo necessária (UTO, 2013). Exemplos de problemas que são particularmente propícios de serem encontrados através da revisão de código fonte incluem problemas de concorrência, falha na lógica de negócios, problemas de controle de acesso e criptografia fraca, bem como *backdoors*[2], *trojans*[3] e outras formas de código malicioso (CONKLIN; ROBINSON, 2016).

2.1.4 Teste de intrusão

Os testes de intrusão têm sido a prática de segurança mais comum e frequente de ser aplicada (MCGRAW, 2006). Do ponto de vista empresarial, segundo Bacudio et al. (2011), os testes de intrusão ajudam a proteger organizações contra falhas:

- Através da prevenção de perdas financeiras;
- Provando a devida diligência e conformidade com os reguladores da indústria, clientes e acionistas;

[2]Recurso utilizado por diversos malwares para garantir acesso remoto ao sistema ou à rede infectada

[3]Programa malicioso que pode entrar em um computador disfarçado como um programa comum e legítimo.

- Preservando a imagem corporativa;
- Racionalizando o investimento na segurança da informação.

Segundo Meucci e Muller (2015), o trabalho envolvido nessa técnica consiste na ação de testar uma aplicação em execução remotamente para encontrar vulnerabilidades de segurança, sem conhecer seu funcionamento interno. Apesar dessa técnica ser eficaz para testar a segurança de redes, isso não ocorre naturalmente quando se trata de aplicações (MEUCCI; MULLER, 2015).

Quando a técnica é realizada em redes e infraestruturas, a maioria do trabalho está envolvido em procurar e, posteriormente, explorar vulnerabilidades conhecidas em tecnologias específicas, pois normalmente são compostas por produtos *off-the-shelf*[4] com configurações padrões (STUDDARD; PINTO, 2011). Como as aplicações web tendem a ser únicas, esse trabalho deve ser mais especializado. Existem ferramentas que foram desenvolvidas para automatizar o processo, mas com a natureza das aplicações web sua efetividade geralmente é baixa (UTO, 2013).

2.2 Metodologia de um teste de intrusão

O objetivo desta seção é apresentar os conceitos envolvidos e a organização de um teste de intrusão, os quais serão relevantes para o desenvolvimento deste trabalho. Aborda-se sobre os conceitos de intrusão, tipos de teste de intrusão, além da especificação das etapas do processo conforme o recomendado pela literatura da área.

2.2.1 Conceitos sobre intrusão

Para um melhor entendimento dos diversos termos relacionados a testes de intrusão utilizados no decorrer deste trabalho, são introduzidos nesta seção os conceitos básicos sobre vulnerabilidade, ameaça e ataque. As definições são dadas pela RFC 2828 - Internet Security Glossary (IETF, 2000).

Vulnerabilidade é definida como uma falha, ou fraqueza, no *design* ou no gerenciamento de um sistema computacional, quando explorada por um atacante, resulta na violação da segurança desse sistema. Tipicamente, uma vulnerabilidade é um ponto de entrada para o sistema.

[4] Produtos de prateleira, soluções prontas adquiridas comercialmente.

Uma ameaça consiste na possibilidade de violação, de forma acidental ou proposital, de uma vulnerabilidade em um sistema computacional causando danos. Os danos são causados por ataques.

Um ataque é o acesso, não autorizado, do sistema computacional. Tipicamente, um ataque ocorre quando um atacante, utilizando-se de uma vulnerabilidade, tenta executar ações maliciosas, como invadir um sistema, acessar informações confidenciais, disparar ataques contra outros computadores ou tornar um serviço inacessível (CERT-BR, 2012). Os ataques podem ser ativos ou passivos e podem ser iniciados de dentro do perímetro de segurança ou do exterior do perímetro de segurança. Em um ataque passivo, tenta-se obter ou fazer uso de informações do sistema. Enquanto em um ataque ativo, existe a tentativa de alterar recursos do sistema ou afetar seu comportamento.

2.2.2 Tipos de teste

Testes de intrusão, de acordo com Uto (2013), podem ser classificados de acordo com a quantidade de informação que é disponibilizada ao testador. Dessa definição, segundo Muniz e Lakhami (2013), as três seguintes terminologias são mais comuns de serem utilizadas para esta classificação:

- **Teste caixa-preta:** visa simular as mesmas condições de um atacante real, que possui acesso somente às informações públicas do alvo, como endereços IPs externos, nomes de domínio e ramo de negócio da entidade.
- **Teste caixa-branca:** nesse tipo de teste, todas as informações do alvo são fornecidas ao testador, incluindo topologia de rede, plataformas utilizadas, diagramas de casos de uso, linguagens empregadas, códigos-fonte e endereçamento interno. O objetivo dessa abordagem é simular ataques que podem ser realizados por pessoas com conhecimento privilegiado do alvo, como funcionários e ex-colaboradores.
- **Teste caixa-cinza:** é uma mistura dos dois anteriores, antecipando ao auditor aquelas informações do alvo que um atacante obteria, cedo ou tarde, em um processo de invasão real. Desse modo, o objetivo dessa modalidade é acelerar a execução do teste, poupando o tempo gasto pelo auditor nos processos de reconhecimento e mapeamento.

Existe também a possibilidade de se incluir mais uma variável na classificação de testes, a metodologia OSSTMM (HERZOG, 2010) considera se a equipe de segurança do

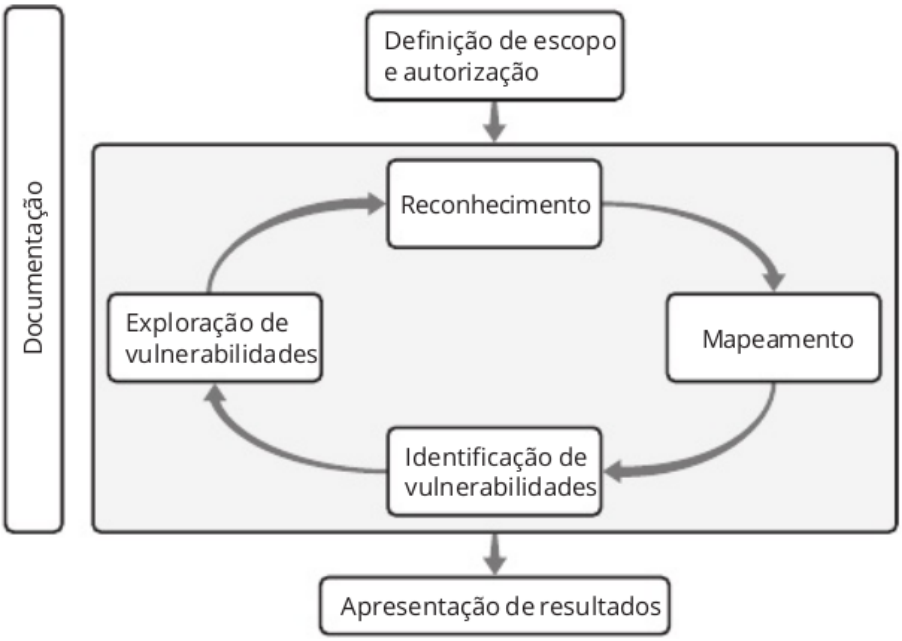
ambiente alvo está consciente da realização dos testes de segurança pelo testador. Com a adição dessa variável, é possível enumerar mais três tipos de testes:

- **Teste duplo-cego:** é um teste caixa-preta, no qual a equipe de segurança do ambiente alvo não é avisada sobre a realização dos testes.
- **Teste duplo-cinza:** é um teste de caixa-cinza em que a equipe de segurança do ambiente alvo sabe o período e escopo dos testes, mas não os vetores ou canais que serão empregados pelo auditor.
- **Teste reverso:** é um teste em que o testador recebe todas as informações disponíveis sobre o alvo e no qual a equipe de segurança do ambiente alvo não é notificada sobre a realização dos testes.

2.2.3 Etapas de um teste de intrusão

Os testes de intrusão são uma ferramenta importante no desenvolvimento e manutenção de uma aplicação segura (MCGRAW, 2006), de forma a evitar que vulnerabilidades sejam inseridas no ambiente produto e avaliar a susceptibilidade dos sistemas a novos tipos de ataque. É fundamental realizar esse processo de acordo com métodos pré-definidos, de modo a permitir que diferentes pessoas alcancem resultados parecidos, que possam ser reproduzidos (UTO, 2013).

Figura 2.1: Etapas de um teste de intrusão



Fonte: (UTO, 2013).

Um teste de intrusão é um processo cíclico que envolve a execução de diversas atividades, conforme o ilustrado na figura 2.1. Nesta seção, as divisões das etapas desse processo, assim como suas definições, são as dadas por Uto (2013), que as adapta ao contexto de aplicações web.

Para se iniciar um teste de intrusão, é necessário se obter do solicitante uma autorização para execução do teste, o escopo que será coberto pela atividade, objetivos e duração dos testes (BACUDIO et al., 2011). Normalmente, isso é feito através de um contrato assinado entre as partes, que além de selar um compromisso, define o tipo do teste, as premissas do trabalho e as informações e privilégios que serão fornecidos pelo contratantes, no caso do teste ser caixa-branca ou cinza (UTO, 2013). Tudo isso ocorre na etapa de definição de escopo e autorização.

Na etapa de reconhecimento, todas as ações são direcionadas a levantar o máximo de informação sobre uma aplicação web que for possível e tentar entender sua lógica de funcionamento (BACUDIO et al., 2011). Normalmente, o auditor busca identificar servidores que suportam a aplicação, sistemas operacionais instalados, linguagens de programação empregadas, nomes e potenciais usuários do sistema, configurações de softwares, entre outros (UTO, 2013). Todas essas informações são obtidas de diversas maneiras, como testes no próprio ambiente ou pesquisas em fontes de informações externas.

Na etapa de mapeamento, todas informações que foram coletadas nas etapas anteriores devem ser relacionadas para se criar um mapa da aplicação, que reflita toda sua estrutura, contendo os arquivos componentes, funcionalidades, tecnologias utilizadas e pontos de entrada de informação (UTO, 2013). Esse processo é realizado se obtendo uma cópia das partes acessíveis da aplicação, para então identificar os pontos de entrada de informação e relacionar o que se foi obtido com as informações de reconhecimento da etapa anterior.

Após se obter um mapa construído da aplicação, o próximo passo consiste em descobrir, caso existam, vulnerabilidades presentes nas camadas que compõem a aplicação. Essas atividades ocorrem na etapa de identificação de vulnerabilidades. O guia de testes do OWASP de Meucci e Muller (2015) expõe uma compilação de verificações para dezenas de vulnerabilidades, agrupadas em classes. Alguns exemplos desses agrupamentos se apresentam como vulnerabilidades relacionadas ao levantamento de informações, gerenciamento de configuração, validação de dados, entre tantos outros. De acordo com Cross et al. (2007), vulnerabilidades encontradas em aplicações web podem ser agrupadas em três famílias:

- Vulnerabilidades presentes em servidores web, que envolve a varredura por vulnerabilidades conhecidas ou, de forma não tão frequente, a descoberta de novas vulnerabilidades nesses servidores;
- Vulnerabilidades devido a *scripts* e páginas padrões expostas, que basicamente são identificadas a partir de varreduras por páginas padrões e *scripts* anteriormente reconhecidos como inseguros presentes em um servidor;
- Vulnerabilidades existentes na aplicação web em si, identificadas a partir do teste de pontos de entrada de informação existentes e das funcionalidades providas pela aplicação.

A identificação de vulnerabilidades se dá através de testes manuais ou automáticos. Para testes automáticos, podem ser utilizados ferramentas como *scanners* de vulnerabilidades, que têm por objetivo encontrar, automaticamente, o maior número de vulnerabilidades existentes em um ativo (UTO, 2013).

Infelizmente, *scanners* de vulnerabilidades em aplicações possuem limitações (DOUPÉ; COVA; VIGNA, 2010). Segundo Studdard e Pinto (2011), tais ferramentas trabalham a partir da análise sintática das respostas da aplicação e identificação de mensagens de erros comuns, códigos de estado HTTP, e conteúdo suprido por usuários sendo copiado para páginas, não sendo capazes de entender o significado semântico do conteúdo analisado. Os *scanners* de vulnerabilidades não conseguem detectar, por exemplo, que a modificação de um parâmetro que determina o preço de um produto pelo usuário é uma vulnerabilidade.

A exploração das vulnerabilidades encontradas, quando possível, é a última fase do ciclo e visa tentar violar a segurança da aplicação (UTO, 2013). O foco dessa etapa é estabelecer acesso a um sistema ou recurso evitando restrições de segurança (PTES, 2014). Normalmente, após a finalização dessa etapa, é esperado que se tenha obtido um maior nível de acesso ao sistema ou uma melhor compreensão dele, de forma a permitir uma abordagem por outro ângulo (UTO, 2013).

2.2.3.1 Documentação e apresentação de resultados

O processo de documentação ocorre paralelamente ao ciclo de testes, sendo empregado desde o contrato formal. Normalmente deve estar incluído na documentação, de forma detalhada, o máximo de informações sobre o ataque, como pré-condições, ferramentas utilizadas e métodos empregados, sendo possível a reprodução da exploração das vulnerabilidades, caso seja desejado por quem solicitou o teste. Comumente é incluído

também recomendações apontando soluções para os problemas encontrados.

De acordo com Muniz e Lakhami (2013), clientes querem saber quão vulneráveis eles são, e os requisitos para corrigir lacunas de modo que o risco de segurança geral de ataque seja reduzida. Conforme Uto (2013), os resultados encontrados devem ser apresentados aos clientes, de forma a serem compreensíveis de acordo com a audiência. Detalhes técnicos das explorações devem ser reservados ao corpo técnico da empresa ou instituição. No caso da audiência ter um perfil mais administrativo, os resultados apresentados devem focar nos impactos resultantes dos possíveis ataques.

2.2.4 Ferramentas

Na realização de um teste de intrusão, é interessante o uso de ferramentas, por existir a possibilidade de elevar a produtividade do processo. Tais ferramentas são classificadas de acordo com suas funcionalidades de forma comum a diversos autores da área como Uto (2013) e Cross et al. (2007), Muniz e Lakhami (2013), existindo a possibilidade de se encaixarem em mais de uma classificação e etapa do processo de testes.

2.2.4.1 Proxies de interceptação

Os *proxies* de interceptação são ferramentas utilizadas para inspecionar requisições e respostas HTTP/HTTPS, permitindo a possibilidade de manipulação de suas informações (STUDDARD; PINTO, 2011). Tais ferramentas normalmente executam localmente, interceptando e exibindo todo conteúdo dos pacotes HTTP/HTTPS gerados pela navegação (UTO, 2013).

Além da funcionalidade básica de interpretação, essas ferramentas podem apresentar diversas funcionalidades como filtros para seleção de mensagens, manutenção de histórico de requisições e respostas interceptados, manipulação das interceptações diretamente do navegador, entre outros (STUDDARD; PINTO, 2011). Exemplos de ferramentas nessa categoria: Paros Proxy (EDGE-SECURITY, 2013), OWASP Zed Attack Proxy (OWASP, 2017) e Burp Suite (PORTSWIGGER, 2017), que é uma suíte de testes paga, mas que disponibiliza seu *proxy* na versão gratuita.

2.2.4.2 Web spiders

Os *web spiders* são ferramentas que automatizam o processo de se obter todo conteúdo navegável de um site (CROSS et al., 2007). Essas ferramentas funcionam a partir da requisição de páginas, fazendo o processamento do seu conteúdo na procura de *links* para acessar outras páginas e continuar o processo recursivamente (STUDDARD; PINTO, 2011).

Além da função básica, *web spiders* podem ainda analisar formulários HTML e submetê-los à aplicação com valores randômicos para expandir a capacidade de navegação entre páginas e fazer o processamento de código JavaScript para extração de URLs (UTO, 2013). Alguns exemplos de *web spiders*, ou de ferramentas com essas funcionalidades, incluem Burp Suite (PORTSWIGGER, 2017) e OWASP ZAP (OWASP, 2017)

2.2.4.3 Scanners de portas e serviços

Os *scanners* de portas são utilizados para determinar que portas em uma estação podem estar sendo escutadas por processos esperando por possíveis conexões (VIVO et al., 1999). Adicionalmente, essas ferramentas são capazes de identificar os serviços específicos que estão escutando essas portas, além de opcionalmente poderem detectar o tipo e versão do sistema operacional utilizado (UTO, 2013). Segundo Vivo et al. (1999), portas abertas podem caracterizar a quantidade de exposição de estações para potenciais ataques.

Essas ferramentas são utilizadas por analistas de segurança para verificar se os ativos zelados estão executando apenas os serviços autorizados, e para levantar informações preliminares do ambiente de teste de intrusão e varredura de vulnerabilidades (UTO, 2013). O software NMAP (LYON, 2016) é um exemplo dessa categoria, sendo capaz de detectar aplicações, serviços e sistemas operacionais (MUNIZ; LAKHAMI, 2013).

2.2.4.4 Scanners e exploradores de vulnerabilidades

As ferramentas dessas categorias podem se concentrar, ou incluir a funcionalidade de detectar vulnerabilidades no servidor web (CROSS et al., 2007). Nesse caso, ações comuns incluem a detecção de arquivos e diretórios instalados por padrão, ou conhecidos como vulneráveis, problemas existentes em uma determinada versão do servidor web e de configuração como Nikto (SULLO; LODGE, 2017) e Webshag (SCRT, 2009). Existem *scanners* para reconhecer vulnerabilidades em plataformas web específicas como CMSs,

para o WordPress[5], por exemplo, existe o WPScan (WPSCAN, 2017).

Para a aplicação web em si, existem ferramentas que possuem a funcionalidade de mapear a aplicação para analisar o conteúdo de páginas e testar pontos de entrada de informação (DOUPÉ; COVA; VIGNA, 2010). Esse tipo de varredura depende, em partes, da qualidade do mapeamento para ser eficiente e é limitada a detectar um conjunto limitado de vulnerabilidades possíveis em aplicações web (UTO, 2013). Algumas ferramentas disponíveis são OWASP ZAP (OWASP, 2017) , Burp Suite (PORTSWIGGER, 2017), w3af (RIANCHO, 2017), Acutinex (ACUTINEX, 2017) e Arachni (LASKOS, 2017).

Existem *scanners* com a função de detectar e identificar vulnerabilidades em sistemas operacionais, aplicações instaladas e demais programas (MUNIZ; LAKHAMI, 2013), complementando a funcionalidade de varredores de portas e serviços. Essas ferramentas se baseiam na busca de vulnerabilidades conhecidas (STUDDARD; PINTO, 2011) e alguns exemplos incluem Nessus (TENABLE, 2017), OpenVAS (GREENBONE, 2017) e a *engine* de *scripting* do NMAP (LYON, 2016).

As ferramentas que possuem a funcionalidade de explorar vulnerabilidades são úteis para determinar o esforço necessário para explorar potenciais vulnerabilidades e julgar o risco associado caso haja um ataque (MUNIZ; LAKHAMI, 2013). Alguns exemplos podem ser o sqlmap (DAMELE; STAMPAR, 2017), fimap (KARIM, 2015) e Beef (ALCORN, 2017) para explorar vulnerabilidades específicas ou ferramentas mais completas como w3af (RIANCHO, 2017) e Metasploit (RAPID7, 2017), que possuem a capacidade de explorar diversos tipos de vulnerabilidades, além de outras funcionalidades.

2.3 Aplicações web

A Web, ou *World Wide Web* como é conhecida, é uma estrutura que permite o acesso a documentos vinculados espalhados por milhões de máquinas na Internet (TANENBAUM, 2011). De maneira técnica, servidores web, navegadores e *proxies*web se comunicam através de trocas de mensagens HTTP (SHKLAR; ROSEN, 2003, p. 65).

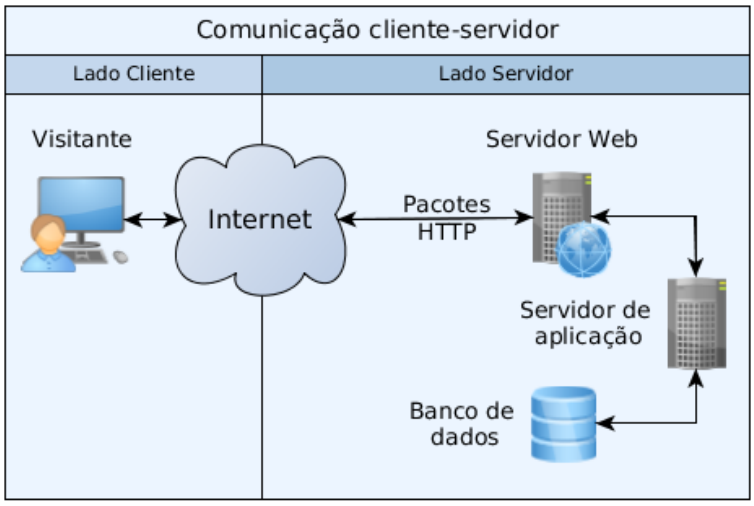
Quando um usuário, através de um navegador, solicita uma página a um servidor web, ela pode ser estática, apenas lendo um documento de um disco, ou dinâmica, resultado de consultas em banco de dados e processamento de um programa (TANENBAUM, 2011). Conforme Lawton (2007), com a evolução da Web, uma mudança importante passou a ser percebida na construção de aplicações web, páginas que apresentavam um

[5]Wordpress CMS, disponível em: http://www.wordpress.com/

comportamento estático passaram a ser interativas e dinâmicas.

Essas aplicações possuem estruturas complexas, como o exemplo ilustrado na figura 2.2. Essa estrutura pode ser visualizada mais facilmente entendendo os papéis do cliente e servidor, além dos componentes e tecnologias envolvidas.

Figura 2.2: Exemplo de funcionamento de um ambiente web



Fonte: Próprio autor.

2.3.1 Funcionalidades *server-side*

Quando um usuário digita uma URL, ou clica em um *link*, o navegador analisa a URL e interpreta a parte entre o protocolo e a barra seguinte, como um nome DNS a ser pesquisado. Munido do endereço IP do servidor web, o navegador estabelece uma conexão TCP. Em seguida, ele envia um comando contendo o restante da URL, que é o caminho até a página nesse servidor. O servidor então retorna o conteúdo para ser exibido.

Um servidor web, em linhas gerais, recebe o nome de um arquivo e retorna seu conteúdo pela rede (TANENBAUM, 2011). Esse conteúdo pode ser estático ou dinâmico, sendo processado por um programa que pode retornar resultados diferentes de acordo com parâmetros recebidos, que podem ser extraídos das seguintes formas:

- Em *query strings* presentes em URLs;
- Em caminho de arquivos de URLs estilo REST;
- Em *cookies* HTTP;
- Do corpo de requisições utilizando o método POST.

Segundo Studdard e Pinto (2011), em adição a essas fontes primárias de entrada, qualquer parte de um pacote HTTP pode ser utilizada por componentes do lado servidor. Para prover funcionalidades *server-side*, aplicações web empregam diversas tecnologias, tais como:

- Servidores web como Microsoft Information Services (ISS), Apache e Nginx;
- *Scripts* que podem ser chamados por servidores web através de plugins para processar requisições, programados em linguagens como PHP[6] e Python[7];
- Plataformas para ASP.NET e Java, que atuam como servidores de aplicação, separados de servidores web, processando somente conteúdo dinâmico;
- Sistemas de armazenamento e demais componentes *back-end* como sistemas de arquivos e serviços de diretório.

Os sistemas de armazenamento em aplicações web são, tipicamente, banco de dados como MySQL e SQL Server, mas também podem ser qualquer fonte que aplicação possa acessar para buscar ou armazenar dados, como arquivos ou saída de comandos (CROSS et al., 2007). A alta gama de tecnologias disponíveis proporciona aos desenvolvedores inúmeras formas de combinar as melhores tecnologias com os objetivos da aplicação.

2.3.2 Funcionalidades *client-side*

Para o lado servidor de aplicações receber e processar entradas e ações de usuários, é necessário que uma interface cliente seja disponibilizada. Devido ao fato de que todas aplicações web são acessadas por navegadores, essas interfaces compartilham de um grupo comum de tecnologias (STUDDARD; PINTO, 2011).

Para permitir a exibição de uma página ao ser acessada, o navegador precisa entender seu formato. Para permitir que todos os navegadores entendam todas as páginas Web, elas são escritas em uma linguagem padronizada chamada HTML (TANENBAUM, 2011). Essa linguagem é baseada em marcações que definem como a estrutura de documentos são renderizadas por navegadores.

JavaScript[8] é uma implementação de ECMAScript, uma linguagem de *scripting* de propósito geral conhecida por estar embutida em diversos navegadores web (ECMA,

[6]Linguagem PHP: https://secure.php.net/
[7]Linguagem Python: https://www.python.org/
[8]Linguagem JavaScript: https://www.javascript.com/

2016). Programas JavaScript são implantados em documentos HTML e interpretados pela maioria dos navegadores (YU et al., 2007). Segundo Studdard e Pinto (2011), JavaScript é utilizada para distribuir o processamento da aplicação entre o cliente e servidor pelos seguintes motivos:

- **Melhorar o desempenho da aplicação:** diversas tarefas podem ser realizadas no lado cliente, evitando requisições desnecessárias ao servidor, como validação de entradas de usuários;
- **Melhorar a usabilidade:** partes das interfaces podem ser atualizadas dinâmicamente sem a necessidade de requisitar uma nova página HTML.

Diversas tecnologias são empregadas para prover as funcionalidades *client-side*, como JSON[9], AJAX[10] e CSS[11]. Entender o papel de linguagens de marcação e de programação em navegadores é o suficiente no contexto deste trabalho.

2.3.3 Sessões e estado

As funcionalidades descritas até então, em conjunto, permitem que aplicações web possam trocar e processar informações entre clientes e servidores de inúmeras formas. Segundo Studdard e Pinto (2011), essas interações estão por trás das diversas funcionalidades providas por aplicações, que necessitam manter informações sobre as ações e dados de um usuário entre múltiplas requisições. Essas requisições são feitas através do uso do HTTP, que é um protocolo de nível de aplicação para sistemas distribuídos, colaborativos e de hipermídia (IETF, 1999) .

Segundo Gutzmann (2001), a natureza "requisição-resposta-desconexão" do HTTP impede que qualquer conexão orientada a estado seja compartilhada entre o cliente e o servidor web, na medida em que esse estado é baseado no próprio protocolo. Devido a essa característica do protocolo, estruturas conhecidas como sessões são mantidas nos servidores, e armazenam dados gerados por usuários entre várias requisições (STUDDARD; PINTO, 2011).

Para conseguir gerenciar sessões, aplicações necessitam estar aptas a reidentificar usuários individuais entre diversas requisições. Normalmente, para resolver esse problema, *cookies* são enviados ao navegador pelo servidor, com o objetivo de lembrar infor-

[9] Uma formatação leve para troca de dados.
[10] Técnica para tornar páginas web mais interativas.
[11] Mecanismo simples para adicionar estilo a páginas web.

mações de um usuário específico (UTO, 2013). Um *cookie* é um elemento formado por uma cadeia de caracteres, usualmente organizado em pares de nome e valor.

2.3.4 Segurança

Aplicações web, por possuírem comportamento dinâmico e interativo, precisam ser capazes de interpretar *scripts*, que podem ser afetados com problemas de segurança devido a programação imprópria, como falta de validação de entradas (STUDDARD; PINTO, 2011). Nos primórdios da Internet, sites não enfrentavam esse tipo de problema em nível de aplicação, visto que proviam somente conteúdo estático e as vulnerabilidades eram encontradas somente em plataformas subjacentes, como sistemas operacionais e servidores web (UTO, 2013).

Segundo a Acutinex (2017), falhas de programação podem levar atacantes a conseguirem acesso a sistemas de armazenamento e consequentemente informações sensíveis ou, de forma menos severa, realizar atos de vandalismo, como *defacements*[12]. Os problemas enfrentados por uma aplicação web são suficientes para exigir atenção a segurança, sendo eles:

- Aplicações web e seus componentes necessitam estar disponíveis o tempo todo para proverem seus *stakeholders*;
- Tecnologias como *firewall* e SSL não garantem proteção para aplicações web pois é necessário que estejam acessíveis publicamente;
- Aplicações web constantemente permitem acesso direto a servidores de banco de dados, o que torna mais difícil de mantê-los seguros. Esses acessos geralmente são feitos através de *scripts* que podem ser inseguros;
- A maioria dessas aplicações são customizadas, ou seja, menos utilizadas e testadas do que softwares de prateleira. Consequentemente, estão mais susceptíveis a ataques;
- Softwares de prateleira presentes em ambientes web como servidores web ou SGBD podem conter vulnerabilidades 0-day[13], estarem desatualizados ou com configurações inseguras vigentes.

[12] Ato de modificar ou danificar a superfície ou aparência de um site.
[13] Vulnerabilidade em um sistema computacional desconhecida pelos seus fabricantes.

3 DESENVOLVIMENTO DA ESTRATÉGIA

Neste capítulo é apresentado a proposta da criação de uma estratégia de avaliação de segurança em sites, se baseando na técnica de teste de intrusão, adaptada para o ambiente da UFRGS, além de um estudo justificando a necessidade de adaptação e os detalhes do seu desenvolvimento. São detalhados as motivações necessárias para a realização do trabalho, além de diversas categorizações e classificações devido a características existentes no ambiente, para fins de planejamento. Também é abordado o processo de implementação, incluindo a avaliação de ferramentas a serem adotadas e, por fim, estatísticas obtidas nesse processo são exibidas e discutidas, com o propósito de visualizar suas implicações na estratégia de testes produzida.

3.1 Contextualização do problema

Com o advento da Política da Segurança da Informação (PSI) (CONSUN, 2014), o DSInf garantiu a autoridade para avaliar se as práticas de segurança da informação estão sendo aplicadas e caso não estejam, poder exigir o alinhamento com a PSI. No contexto deste trabalho, isso possibilitou ao DSInf realizar auditorias de segurança em sites sem a necessidade da autorização de seus responsáveis, assegurando a autoridade de solicitar correções e tomar as medidas cabíveis em caso de não cumprimento.

Essa preocupação existe pois, durante muitos anos, sites das diversas unidades da Universidade foram desenvolvidos e disponibilizados sem medidas que avaliassem a consideração com aspectos de segurança no desenvolvimento. Esse problema era menor quando unidades optavam por hospedar seus sites utilizando o serviço de hospedagem do CPD, onde ao menos a segurança de componentes subjacentes como servidores web, banco de dados e sistemas operacionais era visada. Ainda assim, a necessidade de avaliar a segurança em nível de aplicação persistia.

Para elucidar a gravidade da situação, relatórios como o produzido pela Verizon e USSS (2010), afirmam que, em 2009, a partir de seus registros coletados, identificaram que 92% dos ataques visavam aplicações web, e um relatório mais recente produzido pela WhiteHat (2016), expõe a média de 28 vulnerabilidades encontradas por aplicação web pertencente a instituições de educação, sendo uma das mais altas médias das categorias definidas. Essas estatísticas evidenciam a necessidade de avaliar a segurança dos sites da Universidade. Para realizar essas avaliações, diversas técnicas foram apresentadas no

decorrer do trabalho, no entanto, devido a diversos fatores, somente testes de intrusão podem ser aplicados.

Primeiramente, existe uma dificuldade em se obter os códigos-fontes para realizar análise de código. Caso o site estiver hospedado em servidores das unidades, a possibilitade de se obter o conteúdo necessário é totalmente descartada, devido a resistência dos responsáveis e por, muitas vezes, faltarem pessoas capacitadas a extrair essa informação de um servidor. A justificativa pela inexistência pessoas técnicas em diversas unidades foge do escopo deste trabalho. Mesmo que o site esteja hospedado no CPD, existe o problema de elevar as demandas de trabalho de pessoas que já possuem diversas responsabilidades, no caso os responsáveis pelo serviço de hospedagem. O último ponto a ser considerado é o tempo necessário para a realização da técnica, apesar da eficiência, conforme já apresentado, ante a demanda existente.

As técnicas como modelagem de riscos estruturada a ameaças e inspeção manual também são impraticáveis, devido ainda não ser possível garantir efetivamente a participação de membros do DSInf em fases iniciais de projetos, necessária para eficiência dessas técnicas, além do fato de que a maioria desses projetos já estão concluídos e em produção. Existem iniciativas visando formalizar a validação de segurança em projetos de sites da Universidade, mas essas ações fogem do escopo do trabalho.

Tendo em consideração os dados apresentados, e que a quantidade de sites que necessitam ser avaliados atinge o quarto dígito, é necessário a criação de uma atividade de análise e conformidade de segurança da informação de perfil contínuo. A única forma viável de viabilizar essa ideia, dado a situação apresentada, é através da adaptação do teste de intrusão, apropriado para avaliar a efetividade (ou ineficiência) de medidas de segurança implementadas (BACUDIO et al., 2011), para o ambiente da UFRGS.

3.2 Solução proposta

A solução proposta neste trabalho visa estudar metodologias de teste de intrusão elaboradas por diversos autores da área como Studdard e Pinto (2011), Uto (2013), Cross et al. (2007), Muniz e Lakhami (2013) e Pauli (2013), concomitantemente ao estudo de arcabouços e metodologias para criação de testes de segurança, como as apresentadas por Meucci e Muller (2015) e Herzog (2010) para a criação de uma estratégia de testes customizado dado o contexto retratado, a partir de uma estratégia de desenvolvimento.

A estratégia deve considerar o corpo técnico disponível no DSInf, características

da infraestrutura da rede existente e demais padrões que forem identificados.

Também é visado que testes de intrusão manuais de forma sistemática, que podem ser facilmente reproduzidos por outras pessoas, sejam combinados com testes automatizados, para se aproveitar da natureza repetitiva dos testes e elevar a produtividade. Um estudo conduzido por Austin A. Willians (2011) comprova os benefícios em combinar essas técnicas para o aumento da detectabilidade de falhas de segurança.

3.2.1 Justificativa

O esforço necessário para criação de uma estratégia específica existe devido ao cenário atípico do CPD em relação a empresas e até outras universidades. Os materiais consultados, apesar de auxiliarem em formas de implementar uma estratégia de testes, como dividi-la em etapas, listar vulnerabilidades encontradas em aplicações web e formas manuais e automatizadas de identificá-las, se apresentaram limitados ante aos desafios enfrentados pelo DSInf. Tendo em vista as adaptações necessárias, foram identificadas diversas dificuldades que exigiram soluções independentes da literatura examinada.

Primeiramente, não existem recursos para a contratação de equipes de *pentest* profissionais externas, sendo a atividade delegada aos integrantes do DSInf. Devido às diversas demandas dos funcionários da equipe, a atividade é reservada a bolsistas, que possuem uma frequência de rotatividade muito maior do que a de funcionários. Esse problema, somado a demanda de sites que precisam ser avaliados, motiva a criação de formas de acelerar o aprendizado da técnica para capacitar esses bolsistas, tornando-os aptos a realizarem testes mais rapidamente. A estratégia desenvolvida neste trabalho propõe soluções para o problema.

Os diversos livros sobre pentest não consideram se a empresa ou instituição possuem recursos para adquirir as diversas ferramentas pagas apresentadas, ou se certos requisitos específicos precisam ser cumpridos pela ferramenta para ser utilizável em um ambiente. Parte dos esforços deste trabalho visam formas de avaliar ferramentas que se adequem a requisitos identificados como apropriados para o CPD.

E, finalmente, ao contrário de empresas que contratam serviços de *pentest*, na maioria dos casos, os testes não são solicitados e, muitas vezes, os responsáveis não sabem como proceder com a solução do problema. Essa adversidade tende a variar conforme a unidade contatada, pois as unidades acadêmicas não são homogêneas em relação ao níveil de conhecimento técnico dos responsáveis pela administração dos seus sites e no

comportamento em relação às solicitações de correções, muitas vezes negligenciadas.

Os materiais estudados assumem que um cliente tenha solicitado um serviço de pentest que é finalizado na entrega do relatório com os resultados. Isso não se aplica aos testes feitos no CPD, que após a apresentação dos resultados, é necessário realizar o monitoramento das correções e estipular medidas punitivas em casos de negligência, levando em consideração a importância do site vulnerável. A estratégia desenvolvida prevê formas de lidar com esse problema.

3.2.2 Definições importantes

Analisando as definições de Muniz e Lakhami (2013), em relação ao CPD da UFRGS, responsável pela rede e diversos serviços de TI da Universidade, os testes que utilizam essa estratégia podem ser comparados a testes *caixa-branca*, pois são disponibilizadas informações da topologia da sub-rede da unidade do alvo, sobre seus responsáveis e quando possível, sobre tecnologias empregadas. Conjuntamente a essas vantagens, o fato dos testes partirem de dentro da rede faz que a ação de componentes da segurança de borda, como o *firewall* e IPS/IDS vigentes, sejam evadidas. Em relação aos alvos, segundo as definições complementares de Herzog (2010), esses testes podem ser vistos como *reversos*, pois diversas informações são disponibilizadas e seus responsáveis raramente são avisados. Caso os testes sejam solicitados, podemos considerá-los como *duplo-cinza*.

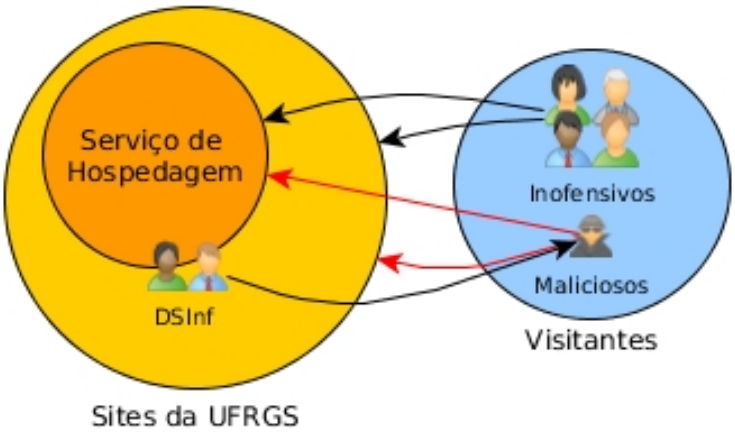
Para fins de organização e direcionamento de esforços, alguns passos sugeridos pela metodologia OSSTMM (HERZOG, 2010, p. 33) para a definição de testes de segurança e gerenciamento de complexidade foram seguidos. Esses passos estabeleceram pontos como:

1. *Definição dos ativos que o processo visa proteger.* Basicamente as ações do DSInf visam proteger os sites da Universidade e os servidores hospedeiros, identificando falhas de segurança e solicitando correções;
2. *Identificação de mecanismos de proteção dos ativos.* Essas informações não podem ser reveladas neste trabalho, mas a possibilidade de haver mecanismos de proteção locais nos alvos é considerada;
3. *Divisão lógica dos ativos dentro de um escopo, definição de vetores de interações internas e externas para idealmente criar testes separados para cada vetor.* Conforme já explicitado neste capítulo, o processo de testes irá considerar a diferença

dos sites presentes no serviço de hospedagem do CPD ou em servidores pertencentes às diversas unidades acadêmicas. Os vetores de interação basicamente representam a interação de visitantes acessando esses sites, sendo que subgrupos desses visitantes, que podem ser usuários internos ou externos, podem ser maliciosos e os integrantes do DSInf visam se passar por eles ao realizar os testes de segurança. A imagem 3.1 ilustra essas interações com o escopo;

4. *Identificação dos equipamentos necessário para os testes.* Basicamente qualquer computador pertencente ao DSInf;
5. *Identificação das informações que serão aprendidas a partir dos testes, no caso se interações com os ativos ou resposta das medidas de segurança ativas serão testadas e como os testes para cada vetor serão definidos.* Essas informações já foram definidas nesta seção. Para cada situação, os testes foram classificados em relação a consciência de sua existência pelo CPD e responsáveis. O processo visa testar essas interações assumindo o papel de possíveis atacantes para então avaliar se as medidas de segurança dos ativos vigentes são suficientes;
6. *Assegurar se o processo de testes de segurança definido está em conformidade com regras de engajamento, viabilizando a execução de um processo de testes sem mal entendidos, equívocos ou falsa expectativas.* Conforme mencionado neste trabalho, a existência da PSI da UFRGS (CONSUN, 2014) viabiliza qualquer processo de teste e análise de conformidade com a segurança da informação.

Figura 3.1: Ilustração dos vetores de interação

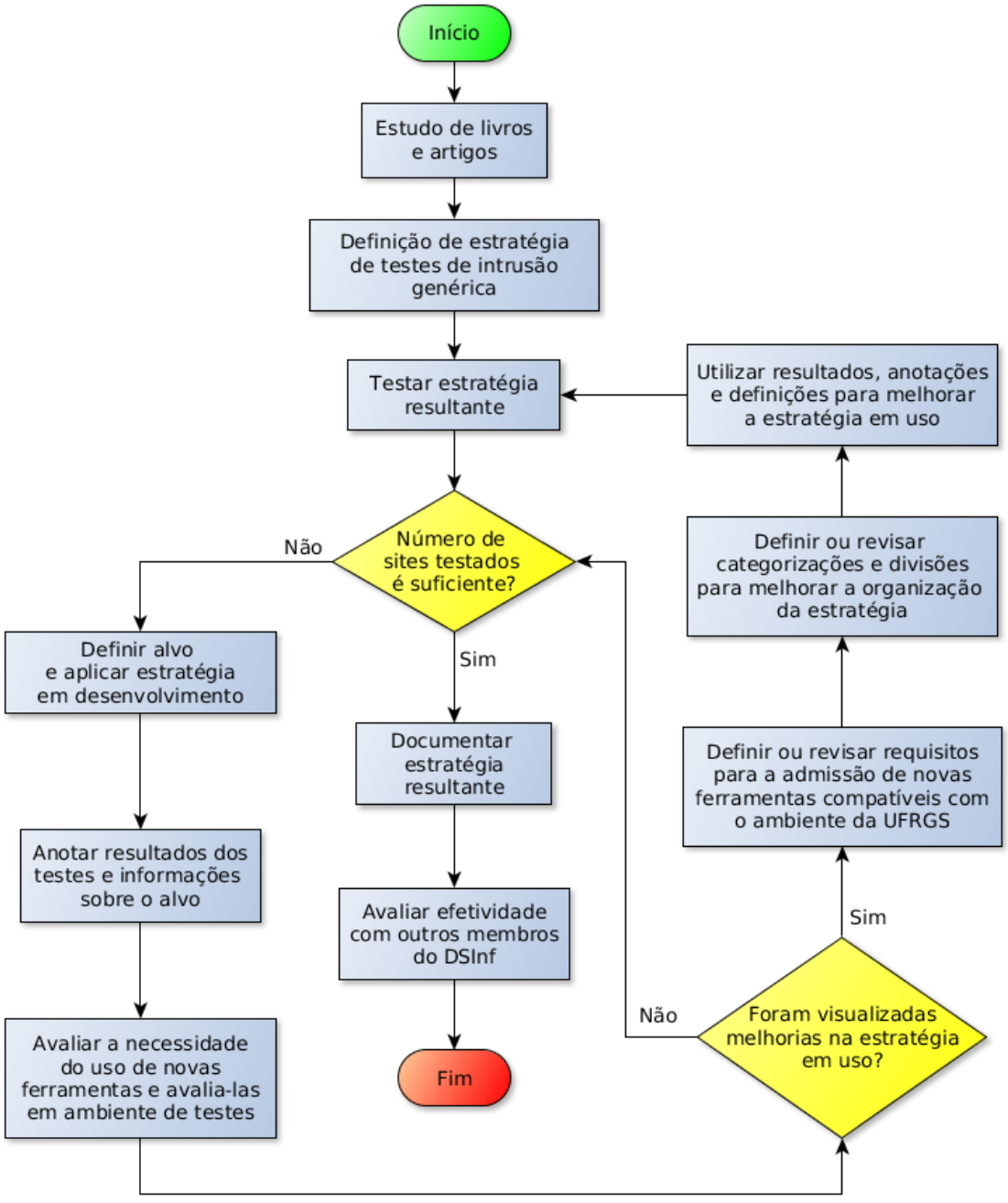


Fonte: O autor.

3.3 Implementação

Para o desenvolvimento da estratégia de testes, dado toda contextualização do problema e necessidade de adequação às peculiaridades do ambiente da UFRGS, foi necessário um processo estruturado, ilustrado na figura 3.2. Como é possível observar, foram executados ciclos de testes, a partir de uma base, que foram seguidos de procedimentos de refinamento baseado em análises empíricas.

Figura 3.2: Fluxograma do processo de desenvolvimento



Fonte: O autor.

Essa base é formada, conforme já detalhado, a partir da consulta de diversos materiais, cujo autores são especialistas em testes de intrusão, com o objetivo de estabelecer um procedimento genérico. Para refinar esse procedimento e cumprir o objetivo deste trabalho, foi definida uma amostra de 50 sites para serem testados.

Para cada site testado, as vulnerabilidades encontradas e diversas informações como tecnologias empregadas e complexidade da aplicação por trás desse site foram anotadas. O objetivo desses registros foi facilitar as decisões em relação a possíveis aperfeiçoamentos a partir dos dados coletados. Concomitantemente aos procedimentos citados, para cada teste, foram identificados problemas, como a descoberta e exploração de determinadas vulnerabilidades ao mapeamento completo de um alvo, que poderiam ser solucionados a partir de soluções automatizadas. Essas soluções se traduziam em ferramentas que foram validadas em ambiente de testes isolados da rede. O ambiente mais utilizado foi o DVWA (DVWA, 2017), uma aplicação propositalmente vulnerável operando em uma estação local. Além de ser possível aprender a identificar vulnerabilidades com esse ambiente, essa ferramenta também foi útil para comprovar a eficiência de boa parte das ferramentas avaliadas.

Para cada ferramenta candidata a ser incluída na estratégia de testes, foi necessário avaliar sua efetividade, tanto em ser útil a resolver o problema da qual foi concebida para solucionar, quanto ao suporte oferecido as necessidades impostas por requisitos definidos no desenvolvimento da estratégia. Para elucidar essa questão, durante as avaliações de exclusão de páginas do escopo de testes (um dos requisitos definidos) em *scanners* de vulnerabilidades, ocorreram problemas de uma ferramenta apresentar essa funcionalidade com defeitos de implementação ou ainda do avaliador se equivocar ao concluir que garantiu que certas páginas não seriam testadas, mas configurou a ferramenta incorretamente.

Caso fosse desejado evitar que qualquer teste fosse feito em um formulário de contato, provavelmente esses erros acarretariam em contas de e-mail lotadas de mensagens contendo indícios de testes de diversas vulnerabilidades, além de, possivelmente, causar instabilidade no servidor de e-mail que contém essas contas. Por possibilidades como as apresentadas, foi necessário que as avaliações fossem realizadas em ambientes isolados, para evitar que terceiros fossem prejudicados.

Após cada iteração, era ponderado, tendo como base os dados coletados e observações empíricas, se possível melhorias poderiam ser realizadas na estratégia. Tópicos como requisitos para ferramentas serem adotadas visando a compatibilidade com as necessidades do DSInf, organização da estratégia, como divisão das etapas e tipos de

vulnerabilidades, e revisão dos procedimentos adotados eram constantemente revisados, incrementados ou redefinidos. O resultado final era uma nova versão da estratégia.

Finalizando os 50 testes previstos, foi considerado finalizado o produto deste trabalho, restando somente documentar, de forma didática e detalhista, incluindo informações confidenciais que não foram expostas neste trabalho, para que outros membros pudessem consultar e aprender a aplciar a estratégia de testes. A avaliação da estratégia desenvolvida, que consistiu em sua aplicação em ambientes de diferentes complexidades por participantes com diferentes níveis de experiência está documentada no capítulo 5, assim como as conclusões obtidas.

3.3.1 Avaliação de ferramentas

Durante a implementação da estratégia de teste de intrusão, diversas ferramentas foram identificadas como úteis. Para incluir as ferramentas no repertório recomendado pelo documento criado após o desenvolvimento da estratégia, uma série de requisitos foram identificados e devem ser seguidos na adição de novas ferramentas:

- **Controle de tráfego:** É interessante que se consiga limitar o tráfego gerado pela ferramenta para não sobrecarregar servidores. Foi verificado que diversas ferramentas implementam opções para definir atrasos entre pacotes, de diversos protocolos, ou entre testes, que podem envolver mais requisições, ou seja, com uma granularidade menos fina;
- **Limitação da cobertura dos testes:** Deve ser possível realizar exclusões dentro dos escopos definidos para as baterias de testes, ou garantir, de forma efetiva, que esse escopo seja respeitado. Para exemplificar, seria muito nocivo que uma ferramenta detectasse uma URL para um site externo e começasse a testá-lo ou que um formulário de inscrição em um evento fosse testado exaustivamente, criando registros com informações estranhas a usuários leigos;
- **Configurabilidade dos testes:** Caso se obtenha informações sobre o ambiente da aplicação testada, é importante existirem opções para limitar os tipos de testes. Por exemplo, caso seja descoberto o sistema operacional do servidor que hospeda o alvo, não é necessário que testes para outros sistemas sejam executados, supondo que essa informação influencie na identificação de determinada vulnerabilidade. Tais medidas acabam por gerar menos tráfego na rede, agilizando os resultados;

- **Mapeamento eficiente:** Se for função da ferramenta, deve ser possível utilizá-la para conseguir identificar todas as páginas de um determinado escopo, assim como todos pontos de entrada de informação;
- **Armazenamento de resultados:** É interessante ser possível armazenar os resultados para futuras consultas, como novas execuções de testes em sites, ou em situações que foram testados e estão em uma mesma sub-rede de alvos anteriores;
- **Verbosidade ajustável:** Se for de interesse, existir uma opção na ferramenta que ajusta o nível de detalhamento das ações executadas pode ser muito útil, principalmente se for desejado aprender o funcionamento da ferramenta e a execução de testes para vulnerabilidades específicas por iniciantes;
- **Recursos didáticos:** Como os bolsistas do DSInf são encarregados de realizar os testes, e o tempo de permanência na equipe é usualmente menor do que o de funcionários, foram procurados por ferramentas que possuíssem recursos para detalhar os resultados encontrados. Por exemplo, alguns *scanners* de vulnerabilidades incluíam informações sobre as vulnerabilidades encontradas, possíveis soluções e *links* para conteúdo relacionado, habilitando um novo meio de aprendizado;
- **Gratuidade:** Não existem recursos para adquirir ferramentas comerciais, então é necessário que as ferramentas adotadas sejam *open-source* ou que dispensam investimentos financeiros para serem utilizadas;
- **Atualizações frequentes:** O fato de um software receber atualizações constantes implica na existência de uma empresa mantendo seu desenvolvimento operante ou de uma comunidade ativa em um projeto de software livre. Tais fatores possibilitam a existência de suporte em caso de dúvidas ou *bugs* encontrados, por exemplo.

No decorrer do desenvolvimento da estratégia, conforme ocorria a identificação de novos requisitos, as ferramentas já adotadas eram revisadas para garantir se ainda havia conformidade com as novas condições de admissão. Caso alguma desconformidade emergisse, a possibilidade de tolerância era ponderada. O resultado desse trabalho pode ser visualizado nas tabelas 3.1 e 3.2, que listam as ferramentas aprovadas e a conformidade com os requisitos definidos. Nas tabelas, um requisito satisfeito, ou recurso existente, é representado pelo caractere "✓", um requisito parcialmente satisfeito pelo caractere "~", e caso o requisito seja dispensável, o caractere "-" é utilizado.

Tabela 3.1: Conformidade das ferramentas adotadas com os requisitos estabelecidos 1

	Ferramentas							
Requisitos	Arachni	Burp Suite	CMSmap	Commix	Droopescan	Fimap	Joomlavs	Joomscan
Controle de tráfego	✓	✓		✓		✓		✓
Limitação da cobertura dos testes	✓	✓	-	-	-	-	-	-
Configurabilidade dos testes	✓	✓	~	✓	~	✓	~	✓
Mapeamento eficiente	✓	✓	-	-	-	-	-	-
Armazenamento dos resultados	✓		✓	✓	✓	✓		
Verbosidade ajustável	✓		✓	✓	~	✓	✓	✓
Recursos didáticos	✓							
Gratuidade	✓	~	✓	✓	✓	✓	✓	✓
Atualizações frequentes	✓	✓		✓	✓		✓	~

Fonte: Próprio autor.

Tabela 3.2: Conformidade das ferramentas adotadas com os requisitos estabelecidos 2

	Ferramentas							
Requisitos	Nikto	NMAP	OpenVAS	OWASP ZAP	Sqlmap	W3af	Wpscan	Xsser
Controle de tráfego	✓	✓	~	✓	✓		✓	✓
Limitação da cobertura dos testes	-	✓	✓	✓	✓	✓	-	-
Configurabilidade dos testes	✓	✓	✓	✓	✓	✓	✓	✓
Mapeamento eficiente	-	-	-	✓	-	✓	-	-
Armazenamento dos resultados	✓	✓	✓	✓	✓	✓	✓	✓
Verbosidade ajustável	✓	✓		✓	✓	✓	✓	✓
Recursos didáticos	~		✓	✓				
Gratuidade	✓		✓	✓	✓	✓	✓	✓
Atualizações frequentes	~	✓	✓	✓	✓	✓	✓	~

Fonte: Próprio autor.

Futuramente, é desejado a definição desses requisitos de forma específica para cada tipo de ferramenta, visando futuras adições no conjunto corrente de forma mais coerente. Por exemplo, a existência de recursos didáticos foi observada em *scanners* de vulnerabilidade, podendo ser um requisito específico de tal categoria, assim como ferramentas desenvolvidas para detectar e explorar vulnerabilidades mais antigas e menos frequentes talvez não precisem mais de atualizações constantes.

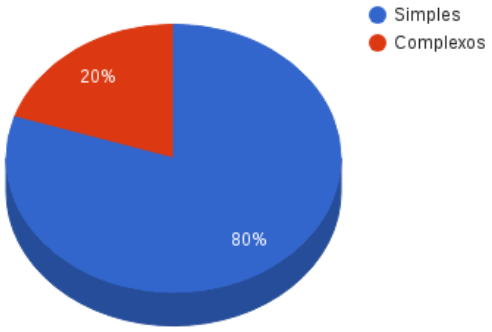
3.3.2 Análise de estatísticas obtidas

Conforme o que já foi explicado neste capítulo, inclusive ilustrado na figura 3.2, diversos dados foram coletadas durante as iterações de testes no desenvolvimento da estratégia, justamente para identificar melhorias e pontos a serem priorizados. Os dados coletados permitiram a criação de estatísticas, cujas aplicações serão demonstradas no capítulo 4.

Uma das preocupações no desenvolvimento era mensurar a distribuição de complexidade dos sites da universidade. Visto isso, para se obter o gráfico da figura 3.3, duas classificações foram definidas:

- **Sites simples:** Sites de notícias e informativos em geral, como páginas de professores, grupos de pesquisa, eventos, entre outros. É considerado a existência de autenticação para interfaces administrativas;
- **Sites complexos:** Aplicações multiusuários, opcionalmente podendo existir interações entre usuários, que armazenem e manipulem volumes consideráveis de informação ou que envolvam transferências monetárias.

Figura 3.3: Complexidade dos sites testados

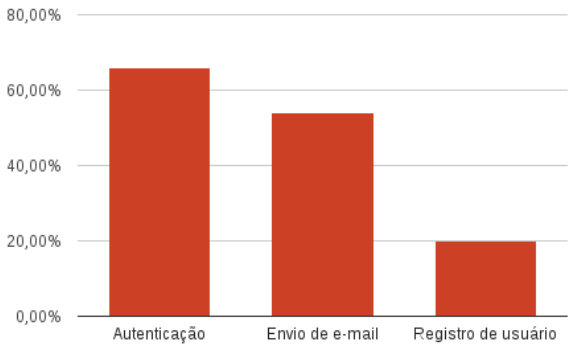


Fonte: Próprio autor.

Foi realizado um mapeamento de funcionalidades mais comuns no sites testados. Assumindo que as proporções presentes na figura 3.4 se repliquem a todos os sites da UFRGS com uma margem de erro baixa, funcionalidades de autenticação, envio de e-mail e cadastro de usuários são as mais encontradas.

Tendo isso em vista, formas específicas de avaliar essas funções foram definidas, com o intuito de localizar falhas específicas que possam facilitar a exploração de certas vulnerabilidades por usuários maliciosos. Adicionalmente, na avaliação dos mecanismos, foi visado auxiliar a visualização da existência de outras vulnerabilidades previstas pela estratégia de testes. Outras funcionalidades comuns a mais de uma aplicação apresentaram baixa presença, não sendo concebível o investimento de tempo na estruturação de testes específicos.

Figura 3.4: Funcionalidades mais frequentes



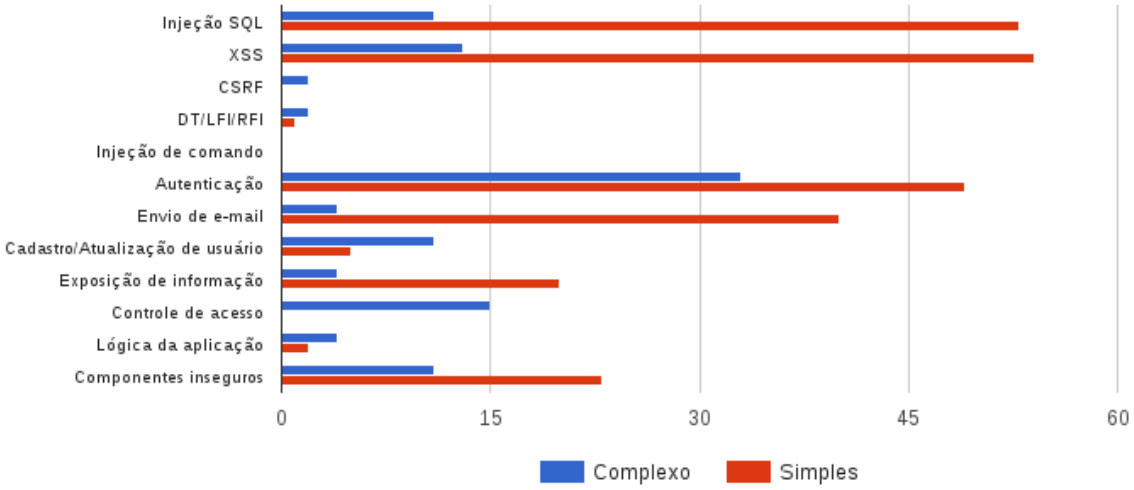
Fonte: Próprio autor.

A execução das 50 iterações resultaram na detecção de 357 vulnerabilidades, uma média de 7,14 vulnerabilidades por site. Dessas vulnerabilidades, 12 categorias de testes foram definidas, sendo algumas uniões de de diversas categorias estabelecidas pelos livros consultados, conforme o conveniente.

A figura 3.5 exibe a quantidade de vulnerabilidades detectadas durante os testes, divididas nas categorias definidas. Conjuntamente é medido a distribuição desses resultados conforme a complexidade dos alvos.

A partir dessas informações, dentre outros fatores, foi possível organizar a divisão de testes de vulnerabilidade em dois grupos de testes distintos, com a pretensão de direcionar o aprendizado de bolsistas. O que também permite a execução da estratégia sem seu total domínio. Essa medida visa permitir que novos integrantes do DSInf assumam a tarefa de avaliar a segurança de sites mais rapidamente, mesmo que inicialmente a efetividade dessas avaliações sejam menores, sendo uma espécie de *trade-off*.

Figura 3.5: Comparação das vulnerabilidades encontradas dado a complexidade do alvo



Fonte: Próprio autor.

Por último, dados como linguagens de programação, software de banco de dados, servidor web e sistemas operacionais também foram coletados, visando mapear as tecnologias mais empregadas na rede da UFRGS. Essas informações auxiliam no direcionamento dos testes previstos pela estratégia e na priorização de tecnologias a serem estudadas. Entender essas tecnologias é um processo gradual que pode auxiliar na compreensão das consequências de vulnerabilidades relacionadas e como corrigi-las, facilitando a documentação de resultados no que tange a sugestões de correção. No entanto, foi escolhido não expor essas informações neste trabalho, por razões de segurança.

4 ESTRATÉGIA DE TESTES

Neste capítulo, é apresentado a estratégia desenvolvida, de forma a detalhar os objetivos definidos, procedimentos e resultados esperados de cada etapa. Essa estratégia é aplicada para cada site pertencente ao escopo do processo de testes de segurança do CPD. A intenção é justificar todas as decisões tomadas, ao invés de apresentar o processo envolvido de forma didática, o que foi reservado ao manual repassado aos outros integrantes do DSInf.

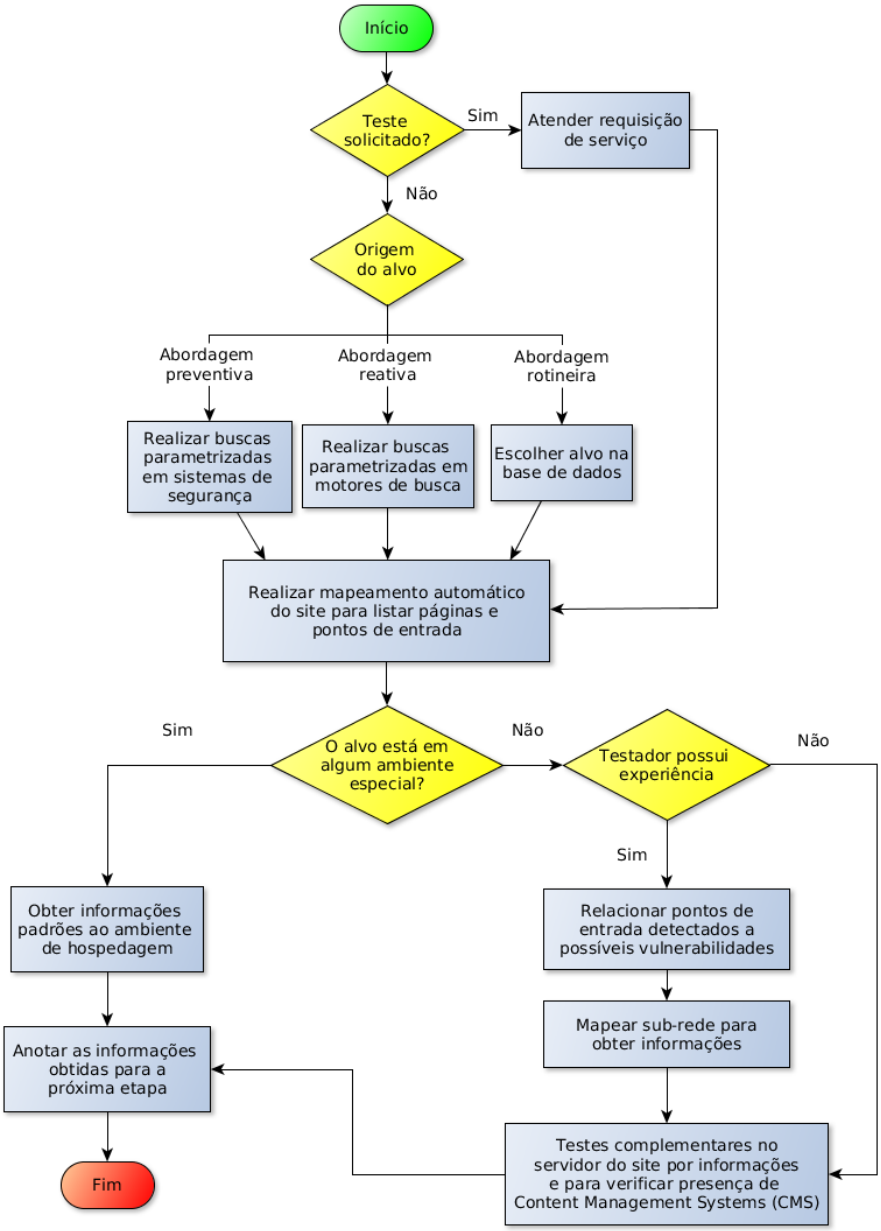
4.1 Identificação e mapeamento de alvos

O objetivo desta etapa é disponibilizar métodos para se obter alvos, para então mapear todas suas páginas e pontos de entrada de informação. Adicionalmente, intenta-se coletar o máximo de informação sobre o ambiente, como tecnologias empregadas nos componentes subjacentes, antes de iniciar os testes de vulnerabilidade. O produto desta etapa é, basicamente, o escopo a ser testado. A figura 4.1 ilustra os passos realizados, que estão descritos nas próximas seções.

Por não existir a necessidade de se obter autorização dos responsáveis pelos alvos para a realização dos testes de intrusão, a etapa de *"definição de escopo e autorização"* é descartada desse processo, sendo a etapa desta seção a primeira a ser realizada. Essa decisão se justifica por não haver necessidade da definição de contratos entre o DSInf e os responsáveis, estabelecimentos prévios do escopo a ser testado, tipo de teste a ser aplicado, informações e privilégios a serem concedidos inicialmente. Esses pontos não diferenciam significativamente de alvo para alvo ou já estão sob posse do DSInf.

Por viés de simplificação, essa etapa é uma união do que autores, como Uto (2013), Studdard e Pinto (2011) e Muniz e Lakhami (2013), costumam dividir em fases distintas de um teste de intrusão ou dedicaram um capítulo inteiro de seus livros para cada assunto, no caso o mapeamento da aplicação e coleta de informações. Essa simplificação se aproveita do fato de que diversas informações que são compartilhadas entre todos os alvos do escopo sob responsabilidade do DSInf já são de conhecimento da equipe. Outro fator que possibilita essa simplificação foi a constatação de que a busca de informações sobre a empresa do alvo (a unidade acadêmica no caso da UFRGS) e seus funcionários, a partir de pesquisa em registros públicos, redes sociais, portais de notícias e ofertas de emprego, demandam muito tempo para pouco benefício.

Figura 4.1: Fluxograma da etapa de identificação e mapeamento de alvos



Fonte: Próprio autor.

Outra diferença da adaptação, em relação a bibliografia consultada, foi priorizar o mapeamento da aplicação e, posteriormente, de forma opcional, realizar o reconhecimento de informações do ambiente. A motivação para tal foi a conclusão que, para iniciantes, os procedimentos adotados para extrair informações do ambiente podem não ser triviais, dando a opção para realizar esses passos conforme se obtém experiência.

A desvantagem imediata dessa abordagem é perder a capacidade de configurar ferramentas de testes, se baseando nessas informações, para serem mais efetivas. Um exemplo seria utilizar o Sqlmap (DAMELE; STAMPAR, 2017), para detecção de injeções

SQL, definindo qual Sistema de Gerenciamento de Banco de Dados (SGBD) a aplicação utiliza. Essa configuração permite que a fase de detecção de SGBD seja ignorada e que testes mais direcionados sejam realizados, diminuindo o tempo de execução do teste e o tráfego gerado na rede. No entanto, foi observado em diversos testes que a falta dessa informação não impediu que a ferramenta obtivesse os mesmos resultados, o que é interessante para iniciantes e os custos dessa lacuna são aceitáveis.

4.1.1 Obtenção de alvos

O início do processo se dá pela definição do alvo a ser testado. Caso os testes não tenham sido solicitados, três formas diferentes de escolha de alvos foram estabelecidas.

Na abordagem preventiva, o *firewall* de borda da rede da Universidade possui capacidades de IPS e gera alertas conforme ataques são detectados se baseando em assinaturas cadastradas. Caso assinaturas relacionadas a ataques em aplicações web sejam selecionadas, é possível visualizar os sites que estão sendo atacados no momento. A partir dessas informações, esta abordagem consiste em justamente testar esses sites para encontrar vulnerabilidades e solicitar correções, antes que os atacantes as explorem.

Na abordagem ofensiva, mecanismos de busca públicos são utilizados para localizar possíveis alvos. Por exemplo, a figura 4.2, que demonstra a busca por alertas MySQL em sites, é um exemplo de busca que poderia ser limitada a sites da UFRGS, visando sites possivelmente vulneráveis a injeção SQL.

Figura 4.2: Um resultado de busca por alertas do MySQL

Fonte: (LANCOR; WORKMAN, 2007).

A ideia por trás disso é tentar visualizar alvos mais propensos de serem atacados através desses meios para então aplicar os testes. A principal técnica utilizada é o *Google Hacking*, que consiste em usar o motor de busca da *Google* para localizar informações sensíveis *online* que deveriam estar protegidas, mas não estão (LANCOR; WORKMAN, 2007). A técnica é utilizada para limitar as buscas aos sites da UFRGS e procurar desde a presença de scripts em sites ou páginas contendo nomes comuns como *"login"* ou *"ad-*

min" até erros padrões de banco de dados ou pilhas de execução de *scripts*. Os resultados dessas buscas retornam sites mais propensos de estarem vulneráveis.

Por último, a abordagem rotineira simplesmente um site não testado presente na lista de sites da Universidade é escolhido para análise. É recomendado a utilização das duas primeiras abordagens sempre que possível, pois ambas expõem sites mais susceptíveis de serem atacados. Caso somente sejam obtidos resultados já testados, então é definido como alvo qualquer site não testado presente em uma lista frequentemente atualizada.

4.1.2 Mapeamento da aplicação

Após a escolha do alvo, todas suas páginas e pontos de entrada de informação devem ser mapeados. Usualmente, parâmetros presentes em URLs de requisições GET, no corpo de requisições POST e em *cookies* são considerados pontos de entrada. Esse procedimento pode ser feito de forma automatizada pela seguintes ferramentas, que se mostraram eficientes:

- **Burp Suite:** Realiza o mapeamento de forma eficiente, com a possibilidade de alterar diversas opções, como o comportamento em relação a formulários e URLs com parâmetros e inserção de atrasos entre requisições, além de possuir uma interface amigável. A desvantagem é o fato de ser paga e suas capacidades de varredura de vulnerabilidades não poderem ser utilizadas;
- **OWASP ZAP:** Semelhante à solução anterior, com exceção de não haver como inserir atrasos na funcionalidade de *"web spider"* (utilizada nessa etapa). Como a ferramenta é gratuita, seus resultados podem ser armazenados e consultados, quando conveniente;
- **W3af e Arachni:** Mais duas opções gratuitas disponíveis que também são utilizadas em outras etapas.

4.1.3 Coleta de informações do ambiente

A metodologia proposta pela PTES (2014) propõe a separação dessa tarefa em níveis de complexidade, definidos pela dificuldade de automação e quanto tempo de trabalho manual é necessário. Definir uma divisão por complexidade nessa subetapa é muito

conveniente para uma estratégia que visa ser utilizável mesmo sem seu total domínio. Mesmo os procedimentos mais simples são opcionais, com apenas uma exceção. Os seguintes procedimentos são considerados de natureza simples:

- **Identificar linguagem de programação:** É uma tarefa usualmente simples de se realizar, baseada na procura por extensões de arquivos pertencentes às linguagens conhecidas nas páginas do alvo;
- **Verificar presença de CMS:** Consiste na procura manual por indícios que indiquem que um site utiliza determinado CMS, ou na utilização de *scanners* de varredura como *CMSmap*, *WPScan* entre outros que também são úteis na detecção dessas aplicações;
- **Identificar servidor web e sistema operacional:** Comandos simples do *GNU/Linux* como *host* e *nc* (Netcat), ou o uso de ferramentas como *NMAP* possibilita a obtenção dessas informações;
- **Verificar se o alvo está em ambiente de hospedagem:** Esse procedimento foi definido como simples, pois, os ambientes disponíveis foram mapeados e instruções de identificação foram definidas para serem facilmente reproduzíveis. É o único procedimento de reconhecimento realmente obrigatório, pois é indesejável realizar testes de alta carga em servidores - ou conjunto de servidores - compartilhados por diversos sites.

De forma complementar, atividades não essenciais mais complexas foram definidas. A não realização dessas atividades não prejudica o andamento dos testes, mas podem ser benéficas, caso sejam postas em práticas por pessoas mais experientes. Os seguintes procedimentos são considerados complexos:

- **Identificar banco de dados:** Frequentemente servidores de banco de dados não estão presentes no mesmo servidor físico da aplicação, ou não estão visíveis devido a regras de *firewalls* locais, mas podem ser descobertos através da injeção de erros ou devido a assunções baseadas em tecnologias comumente associadas detectadas. É uma atividade opcional pois exige experiência, assim como essas informações podem ser obtidas na fase de testes de vulnerabilidade, sendo apenas uma pequena vantagem obtê-las nessa etapa;
- **Associar pontos de entrada de informação ou funcionalidades a possíveis vulnerabilidades:** É uma atividade que custa certo tempo e experiência para ser realizada. Por exemplo, visualizar parâmetros que interagem com um banco de dados

e assumir que possam estar vulneráveis a injeção SQL. Esse procedimento pode auxiliar na organização dos testes, mas não é essencial;

- **Mapear sub-rede do servidor da aplicação:** Esse procedimento pode auxiliar a detectar informações, como SGBD, e qualquer outro serviço em uma sub-rede possivelmente conectados ao alvo. O ferramental exigido é o Sistema de Registro de Estações (SRE), concebido para controlar e identificar dispositivos conectados na rede da UFRGS (CERON et al., 2010), utilizado para identificar a sub-rede de uma estação, e o NMAP, para identificar estações e serviços presentes na sub-rede a partir de varreduras. É necessário certa experiência para executar essas ações, principalmente para analisar os resultados encontrados nessas varreduras e chegar a conclusões em relação ao alvo.

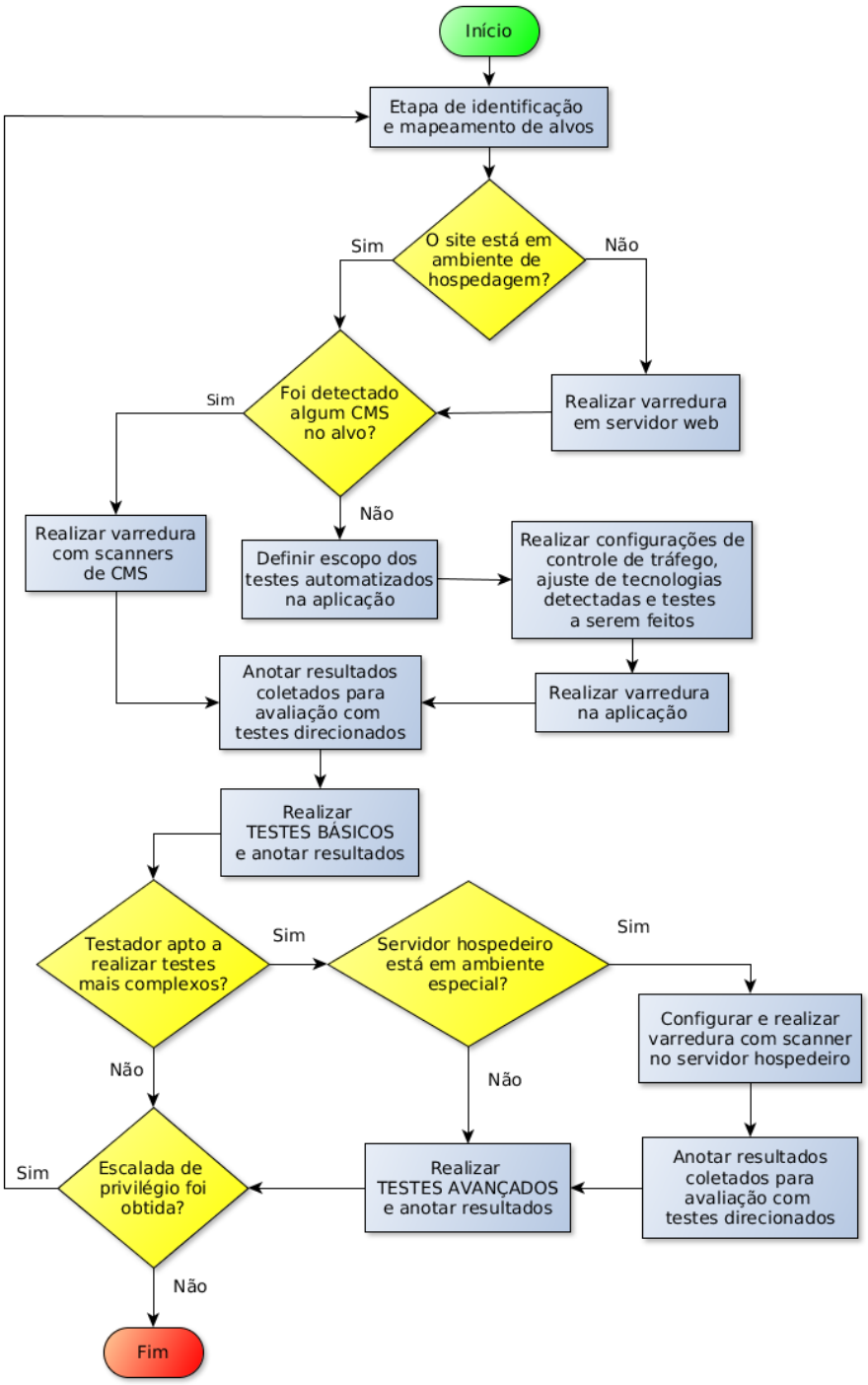
4.2 Identificação e exploração de vulnerabilidades

O objetivo dessa etapa é identificar vulnerabilidades de uma aplicação através de métodos manuais e automatizados, para posteriormente documentá-los e apresentá-los aos responsáveis. As metodologias estudadas categorizam os testes individualmente, ou dividindo-os em grupos de testes que atingem clientes ou servidores, o que não foi adequado ao propósito da estratégia desenvolvida neste trabalho . A divisão empregada na estratégia foi baseada, principalmente, em métricas como complexidade de execução do teste e frequência da vulnerabilidade nos sites da universidade. Esse processo está melhor detalhado nas seções deste capítulo e ilustrado no fluxograma da figura 4.3.

O propósito dessa abordagem é permitir que testes de intrusão sejam executados independentemente das suas etapas estarem totalmente compreendidas, oportunizando que pessoas menos experientes sejam funcionais e que a cobertura dos testes aplicados aumente conforme se adquire experiência. A partir desse princípio, os testes desta etapa foram divididos nas seguintes categorias:

- **Testes básicos:** Visam procurar por vulnerabilidades mais frequentes ou simples de serem testadas, preferencialmente com ferramentas de detecção já desenvolvidas, o que pode auxiliar aprendizes;
- **Testes avancados:** Visam vulnerabilidades que até possam ser frequentes, mas que exigem maior compreensão em relação ao funcionamento de aplicações web, serviços e servidores de rede, podendo necessitar certa responsabilidade nos testes.

Figura 4.3: Fluxograma da etapa de identificação e exploração de vulnerabilidades



Fonte: Próprio autor.

Princípios semelhantes são aplicados as varreduras automatizadas, que também foram definidos divisões se baseando na experiência do analista. O processo de executar as ferramentas de varreduras elegidas é trivial, no entanto, constatou-se a necessidade de senso de responsabilidade para realizar varreduras em servidores e experiência para analisar seus resultados de forma eficiente, por exemplo.

4.2.1 Varreduras automatizadas

De maneira a facilitar a descoberta de pontos vulneráveis na aplicação, componentes subjacentes e servidores físicos, diversos *scanners* de vulnerabilidades foram avaliados e eleitos. Ainda que, segundo DOUPÉ, COVA e VIGNA (2010), esses *scanners* apresentem limitações, não garantindo solidez nos resultados, constatações sobre a utilidade dessas ferramentas, conforme a experiência do analista, foram percebidas.

Primeiramente, foi observado que essas ferramentas podem ser visualizadas como instrumentos de aprendizado de testes de intrusão para iniciantes. Isso é possível devido a possibilidade de todo o processo de automação do mapeamento do escopo, identificação e utilização de vetores de ataque disponíveis poder ser observado e replicado, se desejado. Além disso, seus resultados podem ser interpretados como indicadores, ou suspeitas, de possíveis vulnerabilidades, que podem ser confirmados com testes mais específicos. Consequentemente, auditores, independente da experiência possuída, podem ser mais produtivos utilizando essas ferramentas.

4.2.1.1 Aplicação e componentes

Nessa categoria, foram definidos *scanners* para aplicação web genéricas, CMS e servidores web, que compartilham o fato de poderem ser utilizados por iniciantes sem restrições. Essa decisão se deu através de observações empíricas do uso desses *scanners*, foi verificado que são menos susceptíveis a indisponibilizar servidores devido a altas cargas de testes causadas por configurações insuficientes.

Os *scanners* para servidores web, como *Nikto*, são capazes de realizar testes abrangentes para múltiplos itens, como enumeração de arquivos ou programa potencialmente danosos presentes em servidores, verificação por versões desatualizadas de diversos servidores disponíveis, problemas específicos de versão, detecção de configurações e informações da estação hospedeira (SULLO; LODGE, 2017). Essa ferramenta é utilizada para avaliar servidores web de sites que estejam hospedados em servidores presentes nas unidades acadêmicas, dos quais não há garantia em relação a estarem atualizados e configurados de forma segura. No caso do serviço de hospedagem do CPD, existe essa garantia que componentes existentes estejam em conformidade com as necessidades de segurança.

scanners para CMS, como *CMSmap*, *Joomscan*, *Joomlavs*, *WPScan* e *Droopescan* são utilizados para confirmar a existência ou verificar por problemas de segurança nesses sistemas. Foi observado que essas ferramentas atuam verificando versões desatua-

lizadas desses sistemas e seus *plugins*, enumerando vulnerabilidades existentes para essas versões através de consultas em listas de *Common Vulnerabilities and Exposures* (CVE), enumerando usuários e opcionalmente auxiliando a descobrir senhas fracas, além de detectar configurações inseguras. Varreduras com essas ferramentas são muito úteis devido a diversos sites da Universidade terem sido produzidos utilizando CMSs.

Devido ao fato desses sistemas serem *open-source*, usualmente são mais seguros que sistemas fechados desenvolvidos por grupos limitados de desenvolvedores. Segundo Hoepman e Jacobs (2007), o fato do código-fonte ser disponibilizado, permite que avaliações independentes da exposição de um sistema e do risco associado ao seu uso sejam realizadas, facilitando a correção de *bugs* e incentivando desenvolvedores a se esforçarem mais para obterem mais qualidade em seus códigos. Ainda assim, avaliações por parte do DSInf são necessárias, pois os CMS mais utilizados no ambiente da UFRGS possuem uma base grande de *plugins* disponibilizados, que usualmente possuem muitos *plugins* com problemas de desatualização, de não funcionarem corretamente com a versão atual do CMS, ou por serem inerentemente inseguros por possuírem código de péssima qualidade desatualizados ou herdados (JERKOVIć; VRANEšIć; DADIć, 2016).

Pelos motivos apresentados, caso seja detectada a presença de CMSs em alvos, além do uso de *scanners* de servidores web, foi decidido que somente os *scanners* específicos para esses sistemas sejam utilizados, por serem mais efetivos que *scanners* de aplicações web genéricos nesses sistemas. *scanners* de aplicação web podem ser úteis em todas as situações restantes, basicamente todos os sites desenvolvidos para os propósitos específicos de cada unidade acadêmica, por apresentarem níveis de automação e independência de tecnologias utilizadas em cada ambiente (DOUPÉ; COVA; VIGNA, 2010). Nessa categoria, ferramentas como *OWASP ZAP*, *Arachni* e *W3af* foram selecionadas como satisfatórias dados os requisitos definidos na seção 3.3.1. Caso haja tempo disponível, uma estratégia recomendada é utilizar mais de um *scanner* no mesmo alvo. Esse procedimento pode aumentar a cobertura dos testes automatizados e evidenciar a existência de vulnerabilidades caso diferentes ferramentas apresentem resultados semelhantes.

4.2.1.2 Servidores hospedeiros

Eventualmente, servidores hospedeiros do alvo e de componentes subjacentes, caso não estejam hospedados no mesmo servidor físico da aplicação, poderão ser avaliados através de *scanners* específicos. Para varreduras em servidores serem realizadas, o analista não pode ser iniciante, os servidores não poderão pertencer ao ambiente de

hospedagem, ou executar serviços muito críticos e o alvo não pode ter sido avaliado recentemente.

Apesar da auditoria de segurança em servidores ser realizada pelo DSInf em um processo separado, todo planejamento é definido por pessoas competentes, com senso de responsabilidade, evitando servidores críticos ou em horários de pico. Justamente por isso, é vetado que iniciantes realizem essas auditorias, tanto para diminuir a quantidade de conteúdo inicial a ser aprendido, quanto a ter certeza que tenham desenvolvido certa responsabilidade sobre seus atos. É essencial que a infraestrutura de rede da UFRGS seja compreendida, que servidores que não devem ser avaliados ou que não possam ser avaliados em horários críticos sejam reconhecidos.

No processo de testes de intrusão, a função dessas varreduras é complementar o escopo avaliado por alvo, aumentando a efetividade dessa atividade de segurança. Justamente por existir um processo paralelo com atividades equivalentes, é necessário que o histórico de testes seja revisado, para evitar que servidores testados recentemente sejam testados novamente, dispensando esforço desnecessário.

Para varreduras em servidores hospedeiros, a ferramenta *OpenVAS*, um arcabouço de varredura e gerenciamento de vulnerabilidades (GREENBONE, 2017) foi escolhida. Com essa solução é possível realizar os testes e armazenar os resultados para que outros integrantes da equipe possam consultar. *Nessus*, outra opção pesquisada nessa categoria, é paga, apesar de oferecer algumas funcionalidades por alguns dias sem custos, o que impossibilita sua adoção, portanto foi descartada.

4.2.2 Testes Básicos

Testes que foram definidos como básicos são aqueles que exigem menos experiência para entender suas vulnerabilidades, e que por serem mais populares, possuem ferramentas disponíveis para automatizar a descoberta e exploração de falhas. Pelo fato de existirem ferramentas para maioria desses testes, iniciantes podem acompanhar os passos feitos pelas ferramentas e aprender como funciona o teste de aplicações web. Também foi considerado se as vulnerabilidades testadas foram mais frequentes em sites de complexidade menor, o que não se sustentou como uma métrica definitiva nessa categoria.

Além de ser priorizado o estudo desse conjunto de testes primeiro, é aconselhado o aprendizado dos testes de vulnerabilidades em ordem de frequência no ambiente da UFRGS. Assim, mesmo sem saber aplicar todos os testes dessa categoria, é possível efe-

tuar testes parcialmente efetivos e, futuramente, analisar novamente os mesmos alvos em busca de mais vulnerabilidades. A tabela 4.1, derivada das estatísticas apresentadas no capítulo 3, exibe as vulnerabilidades testadas nessa categoria em relação a quantidade de vulnerabilidades encontradas, em ordem decrescente.

Tabela 4.1: Quantidades de vulnerabilidades encontradas com os testes básicos

Teste	Vulnerabilidades encontradas
Cross-Site Scripting	67
Injeção SQL	64
Informações sensíveis expostas	24
Falhas na lógica da aplicação	6
LFI, RFI e DT	3
Injeção de comandos	0

Fonte: Próprio autor.

4.2.2.1 Cross-Site Scripting

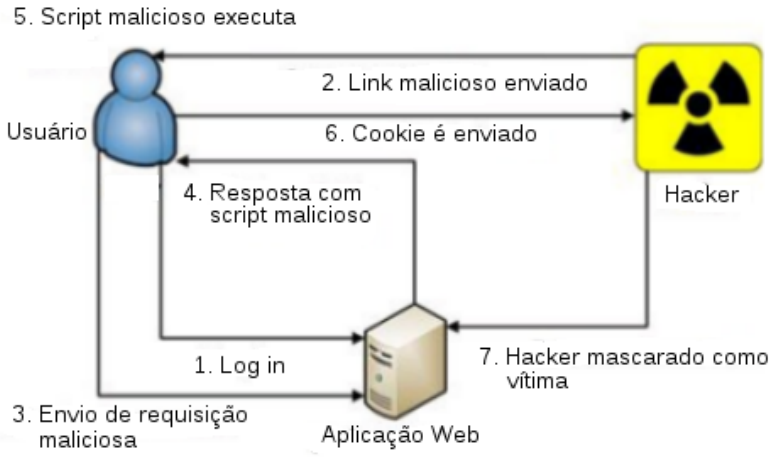
Cross-site scripting (XSS) é uma vulnerabilidade muito difundida em aplicações web modernas (PAULI, 2013). Consiste no abuso da confiança que navegadores possuem a um site. Essa confiança se baseia no fato que todo código de *script* (normalmente escritos em JavaScript) e HTML presentes em uma página serão interpretados e executados pelo navegador.

Basicamente, código de *script* é inserido na saída de uma aplicação que é enviada ao navegador web de um usuário. No navegador, o código, normalmente desenvolvido para propósitos maliciosos, é executado. De forma mais detalhada, a exploração dessa vulnerabilidade costuma ocorrer, segundo (GROSSMAN et al., 2007), de acordo com as seguintes classificações:

- **XSS armazenado:** Costuma ocorrer em aplicações orientadas a comunidades. O atacante submete código malicioso em áreas de sites frequentadas por outros usuários. Essas áreas, normalmente, são páginas de comentários de *blogs*, análises feitas por usuários, tópicos em fóruns e seus *posts*, salas de *chat*, entre outros. Uma vez que o código malicioso for inserido em uma página, será executado toda vez que for acessada, e por isso é a forma mais perigosa dessa vulnerabilidade;

- **XSS refletido:** A partir de qualquer funcionalidade que necessite de informações inseridas pelo usuário e que seja replicada na resposta retornada pelo servidor web, um atacante constrói uma URL especialmente formatada, ou altera pacotes HTTP (no caso de requisições POST), para enviar código malicioso para ser replicado na página, como ilustrado na figura 4.4. Um exemplo comum de funcionalidade que pode ser vulnerável são caixas de busca, que usualmente replicam na resposta de buscas o conteúdo procurado. Como o ataque não fica armazenado em uma página, a exploração é mais difícil, usualmente feita através de engenharia social, enviando e-mails fraudulentos que tentam convencer usuários a acessarem links ou submeter formulários maliciosos para executar o código malicioso;

Figura 4.4: Passos de um XSS refletido



Fonte: (PAULI, 2013)

- **XSS baseado em DOM:** Muito semelhante ao anterior, com a diferença que o código malicioso não é recebido no servidor, pois é diretamente replicado através de código de *script* vulnerável nativo da página, que modifica a Document Object Model (DOM) criada pelo navegador para página.

Atualmente, aplicações web fazem uso extensivo de *script client-side* para melhorar a experiência do usuário, o que fez a popularidade e frequência de ataques XSS aumentar (VOGT et al., 2007). Isso se provou verdade nos sites da Universidade, visto que foi a vulnerabilidade mais detectada.

XSS foi incluído nos testes básicos por ser uma vulnerabilidade de fácil detecção (OWASP, 2013) e existir uma ferramenta eficiente desenvolvida, além da alta frequência de detecção, independente da complexidade dos alvos. A Xsser (XSSER, 2017) é uma ferramenta que cumpre boa parte dos requisitos estabelecidos e pode auxiliar, principal-

mente, iniciantes aprenderem sobre XSS e formas de detecção. O SQLmap, para detecção de injeção SQL, possui mecanismos básicos para detecção de suspeitas de XSS, e pode ser utilizado também.

4.2.2.2 Injeção SQL

Esse tipo de ataque tem como alvo bancos de dados que são acessíveis via aplicações web e se aproveita de falhas na sanitização de entradas em componentes web como scripts *server-side* (BOYD; KEROMYTIS, 2004). De forma mais específica, essa vulnerabilidade consiste na inserção de comandos arbitrários, através de parâmetros vulneráveis de uma aplicação, para uma base de dados via SQL.

As consequências da exploração dessa vulnerabilidade são graves, visto que permitem aos atacantes obterem acesso irrestrito a bancos de dados expondo informações potencialmente sensíveis (HALFOND; VIEGAS; ORSO, 2006). Diversas formas de ataque de injeção SQL são bem conhecidas na literatura, tais quais:

- **Baseado em tautologia:** O objetivo desse ataque é injetar código em uma ou mais declarações para que sempre sejam validadas como verdadeiras. O uso mais comum deste ataque é para a quebra de mecanismos de autenticação (HALFOND; VIEGAS; ORSO, 2006);
- **Consultas ilegais ou de lógica incorreta:** Um atacante, a partir de consultas incorretas e os erros retornados, pode obter diversas informações sobre o sistema, como estruturas de banco de dados;
- **Consultas com operador UNION:** Através de um parâmetro vulnerável, um atacante pode obter dados de uma tabela diferente da que o desenvolvedor definiu que o parâmetro interagisse originalmente;
- **Consultas adicionais:** Se possível, nessa técnica tenta-se adicionar múltiplas consultas através de um parâmetro vulnerável;
- **Execução de procedimentos armazenados:** Esse ataque visa executar procedimentos armazenados em um banco de dados. Atualmente, a maioria dos vendedores disponibilizam seus SGBDs com conjuntos de procedimentos armazenados para estender as funcionalidade dos bancos de dados e permitir interações com o sistema operacional (HALFOND; VIEGAS; ORSO, 2006);
- **Inferência:** Nessa forma de ataque, uma consulta é modificada para adivinhar valores de dados em banco de dados de forma a disparar ações de acordo com a res-

posta. Esse ataque é utilizado quando é sabido a existência da vulnerabilidade em um parâmetro, mas a aplicação exibe um erro genérico como medida de segurança. Uma forma comum de inferência se baseia injeção de atrasos na consulta caso a resposta seja verdadeira ou falsa, sendo possível descobrir a existência de um valor de acordo com o tempo que a aplicação demora para responder a requisição.

Nos sites testados, injeções SQL foram frequentemente descobertas e o SQLmap se mostrou uma ferramenta extremamente eficiente, possuindo compatibilidade com todos os SGBDs detectados e conseguindo automatizar a detecção e exploração em todos os casos. Devido a essas constatações, testes para descobrir aplicações vulneráveis à injeção SQL foram incluídas na categoria de testes básicos por ser uma vulnerabilidade grave e frequente no ambiente, ainda mais em sites com funcionalidades mais simples. O que também influenciou a decisão foi o fato do SQLmap conseguir automatizar todos os passos necessários, o que pode ser muito útil para iniciantes que talvez não tenham noções sólidas de SQL e do funcionamento de SGBDs.

4.2.2.3 Informações desnecessariamente expostas

Nas verificações por informações desnecessariamente expostas é visado a busca por problemas em relação às informações que possam auxiliar atacantes em suas tentativas de intrusão e não deveriam estar disponíveis. Para a formalização desse teste, basicamente foi feita a compilação de diversos testes presentes nos materiais consultados.

O objetivo principal dessa junção foi procurar diminuir a complexidade do processo reduzindo a segmentação das categorias de teste. Diversos problemas encontrados nos sites da universidade apresentavam consequências semelhantes, mas não eram complexos o suficiente para justificar mais seccionamentos. Portanto, como forma de organização, as seguintes verificações estão definidas nesta seção:

- Identificação de versões antigas de sites no ar ou seções não utilizadas;
- Detecção de código *server-side* comentado no código fonte das páginas;
- Detecção de páginas padrões de componentes como servidores web, que usualmente não são manipuladas pela aplicação. Páginas não referenciadas, ou esquecidas, podem ser usadas para se obter informações importantes de uma infraestrutura ou credenciais (MEUCCI; MULLER, 2015, p. 54);
- Detecção de erros na aplicação, como erros padrões de banco de dados ou pilhas de execução de *scripts*.

Para a detecção de boa parte desses problemas, a análise dos resultados dos *scanners* de varreduras se mostrou eficiente, aos menos para os três últimos itens. Devido a existência de meios automatizados para auxiliar na descoberta dessas falhas, com o adendo que, a partir das estatísticas obtidas, percebeu-se uma maior incidência em sites menos complexos, foi decidido incluir esses testes na categoria de testes básicos.

Outra forma de detecção consiste na análise passiva da aplicação enquanto é realizado seu mapeamento e teste por vulnerabilidades. Exemplos frequentes dos resultados dessas análises seriam erros de aplicação que surgem através de testes, ou a descoberta de seções antigas abandonadas de um site durante seu mapeamento, tendo como exemplo páginas reservadas a eventos acadêmicos ocorridos há anos, com funcionalidades como registros de usuário ativas.

4.2.2.4 Falhas na lógica da aplicação

Toda aplicação web, assim como qualquer programa de computador, emprega lógica para disponibilizar suas funcionalidades, quebrando processos complexos em pequenos passos discretos. Desenvolver essa lógica de maneira segura é um problema vigente em aplicações web, visto que estão muito mais expostas a usuários maliciosos do que programas executados em uma única estação sem comunicação com a Internet.

Testes para identificação de falhas na lógica tentam reconhecer possíveis defeitos no código da aplicação para avaliar se suas existências implicam em vulnerabilidades de segurança. Diferente de vulnerabilidades mais comuns como injeção SQL e XSS, que possuem assinaturas facilmente reconhecíveis e vetores de exploração mais estudados (STUDDARD; PINTO, 2011), não existem ferramentas efetivas para detectar esse tipo de falha, sendo a detecção limitada a procedimentos manuais.

Apesar dos testes evidenciarem incidência maior dessa falha em sites mais complexos, foi listado , no manual reservado aos integrantes do TRI, diversos procedimentos isolados para identificação de casos de teste recorrentes, de fácil execução e análise de possíveis problemas de segurança. O objetivo da documentação desses procedimentos é tentar auxiliar iniciantes a desenvolverem um raciocínio voltado a visualização de vulnerabilidades de segurança em qualquer interação disponibilizada por aplicações. Por esses motivos, testes de falhas na lógica foram incluídos nos testes básicos, sendo alguns exemplos:

- Testar a possibilidade de inserir mais caracteres do que um campo em algum formulário supostamente permite;
- Inserir valores inteiros maiores do que o esperado em parâmetros;
- Testar se a aplicação valida valores inseridos diferentes do esperado (valores diferentes de “m” ou “f” no campo “sexo” de um formulário de inscrição);
- Testar a possibilidade de alterar valores de parâmetros que não poderiam ser modificados;
- Testar a submissão de arquivos com extensões diferentes das esperadas;
- Testar o comportamento da aplicação submetendo parâmetros sem valor;

4.2.2.5 Inclusão de arquivos e passagem de diretórios

Muitas aplicações web utilizam e gerenciam arquivos e diretórios (locais ou externos) como parte de suas funcionalidades. Vulnerabilidades podem vir a existir caso mecanismos que manipulem referências a arquivos sejam expostos a usuários e não possuam métodos de validação de entradas eficientes. Com o mesmo propósito do teste anterior, as seguintes vulnerabilidades, com semelhanças na forma de detecção e exploração, foram compiladas em uma única seção:

- **Local File Inclusion (LFI):** Consiste em conseguir injetar na página vulnerável arquivos contidos no servidor hospedeiro da aplicação e executá-los;
- **Remote File Inclusion (RFI):** Possibilita que arquivos de fontes externas sejam incluídos em entradas da aplicação indevidamente;
- **Directory Traversal (DT):** Semelhante a LFI, com a diferença que se consegue somente ler o conteúdo de arquivos presentes no sistema e não executá-los na aplicação.

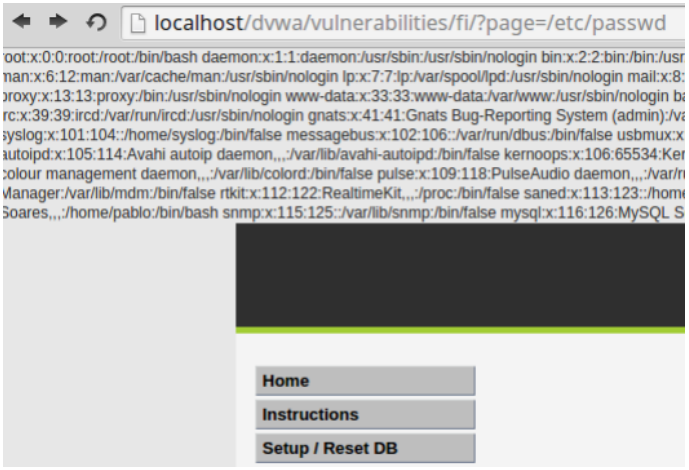
De acordo com Hubczyk, Domanski e Domanska (2012), alguns sites costumam prover navegação entre páginas através da inclusão de arquivos em parâmetros armazenadas em URLs, que podem ser manipulados de forma maliciosa. Um exemplo disso seria o parâmetro “site” em <*http://example.com/index.php?site=home.html*>, cujo valor atribuído é utilizado pela aplicação para localizar um arquivo cujo nome seja igual a esse valor no sistema de arquivos local. Apesar do baixo número de detecções dessas vulnerabilidades, durante o desenvolvimento da estratégia, diversos sites proviam recursos de navegação através da inclusão de arquivos em parâmetros de URLs.

Devido a frequência de mecanismos em sites da Universidade que possam estar vulneráveis a esses ataques, e a existência de uma ferramenta específica para detecção de inclusão de arquivos, foi decidido incluir essas vulnerabilidades na categoria de testes básicos. A ferramenta elegida, *Fimap*, possui funcionalidades de detecção e exploração dessas vulnerabilidades, sendo capaz de aplicar vetores de ataque para mais de um sistema operacional hospedeiro. Essa função pode auxiliar iniciantes que não possuam conhecimentos suficientes da estrutura de diretórios de sistemas operacionais variados, essencial para tentativas de se obter arquivos com informações sensíveis de sistemas em casos de LFI e DT, por exemplo.

4.2.2.6 Injeção de comandos

Aplicações web podem utilizar de interações com o sistema operacional, através de execução de comandos em nível de sistema, para prover funcionalidades. A injeção de comandos ocorre quando entradas de usuários são concatenadas a comandos maliciosos sem a devida validação, como ocorre na figura 4.5 o que pode resultar em consequências graves. Normalmente, atacantes utilizam essas falhas para criar, deletar ou alterar permissões de usuários em um sistema ou para extrair o máximo de informação possível de um sistema (PAULI, 2013).

Figura 4.5: Exemplo de exploração de parâmetro vulnerável a LFI



Fonte: Próprio autor.

Apesar de não ter sido detectado a presença dessa vulnerabilidade nos sites testados, devido a sua gravidade, a quantidade de sites que ainda precisam ser avaliados e a existência de uma ferramenta para automatizar o teste, a vulnerabilidade foi incluída nos testes básicos. A ferramenta *Commix* possibilita a automatização da descoberta e explora-

ção de injeção de comandos em diversos sistemas operacionais, exibindo todos os passos executados para iniciantes que talvez não dominem o uso dos diversos interpretadores de comando disponíveis, como o *bash* no *GNU/Linux* ou *cmd.exe* no *Windows*.

4.2.3 Testes avançados

Esta categoria reúne vulnerabilidades mais complexas de testar ou que exigem mais experiência e conhecimento para entendê-las e realizar testes. A maioria dos testes não possuem ferramentas disponíveis para auxiliar na identificação de vulnerabilidades.

Da mesma forma que é indicado nos testes básicos, é recomendado que o aprendizado dos testes siga uma ordem priorizando vulnerabilidades mais frequentes. A tabela 4.2 exibe as vulnerabilidades desta categoria em ordem decrescente de resultados encontrados.

Tabela 4.2: Quantidades de vulnerabilidades encontradas com os testes avançados

Teste	Vulnerabilidades encontradas
Mecanismos de autenticação	84
Interações com envio de e-mails	44
Componentes inseguros ou em desuso	34
Registro e atualização de dados	16
Mecanismos de controle de acesso	15
Cross-Site Request Forgery (CSRF)	2

Fonte: Próprio autor.

Outro objetivo dos testes dessa categoria é auxiliar na identificação de vulnerabilidades mais simples dentro do escopo testado, assim como orientar a avaliar os resultados obtidos na exploração dessas vulnerabilidades de forma a identificar mais falhas de segurança. A ideia por trás dessa complementação aos testes mais básicos é não sobrecarregar iniciantes, ensinando-os a auditar sites de forma gradual.

Algumas vulnerabilidades testadas se mostraram muito frequentes mesmo em sites mais simples. Ainda assim, conclusões como o fato de exigirem mais capacidade analítica para serem compreendidas, não haver ferramentas disponíveis e não serem tão severas influenciaram a inclusão de seus testes nessa categoria. Outro fator levado em conta foi a

possibilidade de testes para essas vulnerabilidades poderem comprometer todo ambiente caso sejam realizados por pessoas inexperientes.

4.2.3.1 Mecanismos de autenticação

No contexto de segurança computacional, autenticação é o processo de tentar verificar a identidade digital do remetente de uma comunicação (MEUCCI; MULLER, 2015). À primeira vista, a autenticação é conceitualmente um dos mais simples mecanismos de segurança utilizados em aplicações web (STUDDARD; PINTO, 2011). Os desenvolvedores frequentemente implementam a autenticação e gerenciamento de sessão em suas aplicações de forma personalizada, mas a implementação correta é difícil (OWASP, 2013).

Diversos sites da UFRGS possuem formulários de *login*, cujos testes demonstraram a necessidade de avaliá-los, conforme a quantidade de falhas detectadas durante o desenvolvimento da estratégia. Para resolver esses problemas recorrentes e identificar falhas em sistemas de *login* e armazenamento de senhas, uma compilação de verificações foi definida, representada pelos seguintes itens:

- Verificar se as informações de autenticação estão sendo transmitidas por canais criptografados. De forma mais específica, avaliar a existência de HTTPS ou qualquer outra forma de cifragem de dados sensíveis;
- Verificar se existe possibilidade de ataques de força bruta. Por exemplo, verificar presença de *CAPTCHA* no formulário de autenticação;
- Verificar possibilidade de enumeração de usuários. Uma forma muito comum é executando tentativas de autenticação propositalmente falhas com usuários válidos e inválidos e verificar se aplicação responde de forma diferente aos testes;
- Verificar tecnologias envolvidas na autenticação e testar vulnerabilidades relacionadas. Condições como a existência de injeção SQL podem resultar no contorno da necessidade de autenticação (CROSS et al., 2007);
- Na existência de injeção SQL, em qualquer funcionalidade de uma aplicação, verificar se a visibilidade de tabelas que armazenam credenciais de acesso de usuário. Distinguir se as senhas são armazenadas de maneira cifrada ou em texto claro;
- Avaliar a presença de informações sensíveis em parâmetros GET. Informações transmitidas por GET podem ficar armazenadas em *logs* de servidores web e histórico de navegadores, o que pode criar problemas de segurança (MEUCCI; MULLER, 2015);

- Avaliar parâmetros ocultos (que não são diretamente visíveis a usuários) na autenticação e testar por falhas de segurança;
- Verificar mecanismos de recuperação de senha para descobrir se as senhas estão sendo armazenadas em texto claro;
- Testar por credenciais simples ou padrões. Casos frequentes seriam *"admin/password"*, *"admin/admin"* ou credenciais padrões de softwares conhecidos;
- Verificar necessidade de VPN. Mecanismos de autenticação específicos de interfaces de administração importantes deveriam utilizar a VPN da UFRGS para adicionar uma camada de proteção;
- Verificar se formulários de *login* estão vulneráveis a CSRF e avaliar necessidade de correção conforme a importância do site para a Universidade.

Apesar de ser a vulnerabilidade mais frequente, principalmente em sites simples, realizar essas auditorias envolvem conhecimentos muito amplos, não somente de aplicações web, sistemas subjacentes e redes, mas também do ambiente tecnológico da Universidade, o que pode sobrecarregar analistas em processo de aprendizagem. Por tais razões, além de não existir ferramentas para auxiliar na detecção de problemas, avaliações em mecanismos de autenticação foram categorizadas como testes avançados.

4.2.3.2 Interações com envio de e-mails

Muitas aplicações web contém funcionalidades para usuários submeterem mensagem pela aplicação, usualmente implementadas interfaceando com servidores de e-mail (ou SMTP). Se um atacante poder enviar uma entrada adequada, que não seja filtrada ou desinfectada, ele pode ser capaz de injetar comandos SMTP arbitrários nessa comunicação (STUDDARD; PINTO, 2011). Usualmente é permitido que o conteúdo da mensagem e o endereço de e-mail do usuário sejam especificados. Todavia, qualquer campo sobre o controle do usuário pode estar vulnerável à injeções SMTP.

Frequentemente, sites da Universidade possuem formulários de contato, ou de inscrição em eventos que, ao serem submetidos, fazem o envio de mensagens com as informações registradas para servidores de e-mail pré-determinados. É de suma importância avaliar a segurança da implementação dessas funcionalidades, cujos problemas não se restringem somente às injeções SMTP.

Caso as falhas possíveis sejam exploradas de forma combinada, um indivíduo ma-

licioso pode espalhar *phishings*[1] utilizando os servidores de e-mail da Universidade, além de poder prejudicar a disponibilidade desses servidores. Para identificar implementações vulneráveis, normalmente as seguintes avaliações são realizadas:

- Verificar a necessidade das mensagens serem transmitidas através de canais criptografados conforme o sigilo necessário do tipo de informação envolvida na comunicação;
- Verificar se existem mecanismos de proteção contra ataques de força bruta, como *CAPTCHAs*, para impedir que múltiplas mensagens sejam enviadas e prejudiquem a disponibilidade de servidores ou que *phishings* possam ser espalhados caso o formulário de e-mail testado também seja vulnerável à injeção SQL;
- Avaliar a possibilidade de enviar e-mails para endereços arbitrários a partir da modificação de parâmetros na submissão do formulário (injeção SMTP);
- Avaliar possíveis parâmetros ocultos por implicações de segurança. Pode ocorrer de existirem parâmetros ocultos que armazenem o endereço de destino da mensagem, possibilitando o envio de e-mails para endereços arbitrários sem a necessidade de injeções SMTP.

A decisão de categorizar esses testes como avançados é, principalmente, a falta de uma ferramenta específica para avaliar interações com envio de e-mails. Outro fator influente, observado no desenvolvimento da estratégia, é o risco do aprendiz, sem conhecimentos suficiente da infraestrutura por trás dos sites da Universidade, utilizar ferramentas automatizadas erradas para testar os parâmetros do formulário. Devido ao risco de gerar condições de negação de serviço pelos testes automatizados, foi definido que iniciantes sejam desencorajados a testar páginas contendo essas funcionalidades, justificando mais um motivo de ser considerado um teste avançado.

4.2.3.3 Componentes inseguros ou em desuso

A força da cadeia que compõe uma aplicação web é tão forte quanto seu elo mais fraco (MEUCCI; MULLER, 2015). Portanto, procurar entender os riscos que configurações inseguras e componentes desatualizados podem apresentar são tão importantes quanto às vulnerabilidades na aplicação em si. Os seguintes componentes usualmente são considerados neste teste:

[1] No contexto de e-mails, é um golpe que tem o objetivo de obter informações e dados pessoais importantes através de mensagens falsas

- *Scripts* feitos por terceiros utilizados no site alvo;
- Aplicações web de prateleira como CMSs e interfaces de gerenciamento de dispositivos de rede, banco de dados, entre outros;
- Softwares como servidores web e de banco de dados utilizados pela aplicação;
- Sistemas operacionais e demais serviços instalados nos servidores que hospedam o site alvo e seus componentes.

Existem bases de dados que associam vulnerabilidades descobertas em softwares, informando as versões vulneráveis e armazenando muita informação útil. Esses serviços usualmente costumam classificar vulnerabilidades de acordo com o *Common Vulnerability Score System* (CVSS), um arcabouço aberto para comunicação das características e gravidade das vulnerabilidades de software (FIRST, 2015), além de indicar exploits para provas de conceito. *CVE*[2], *CVE Details*[3], *Exploit-db*[4] e *NVD*[5] são exemplos de bases de dados consultadas.

Em uma verificação de segurança, é necessário detectar componentes desatualizados que possuam vulnerabilidades, analisando o risco de mantê-los operacionais para avaliar a necessidade de atualização desses componentes. Também é útil tentar detectar componentes abandonados, para que sejam removidos do ambiente e diminuir a superfície de ataque existente. Além de falhas nos componentes, a forma que estão configurados também deve ser avaliada. Os seguintes exemplos de configurações inseguras em componentes são frequentemente avaliados:

- A partir de injeções SQL detectadas, de acordo com diversas boas práticas de segurança de banco de dados (Berkeley, 2017), verificar problemas como o usuário utilizado pela aplicação web para se conectar ao SGBD não é administrador; verificar se o SGBD é acessível somente pelos servidores que o utilizam; verificar se a aplicação alvo possui acesso restringido a tabelas sob posse, evitando que atacantes acessem informações reservadas a outras aplicações e serviços; verificar se a conta utilizada pela aplicação possui somente as permissões (leitura, escrita, operações específicas) necessárias;
- A partir de injeções de comando, verificar se o usuário utilizado pelo servidor web não é administrador do sistema, para evitar o controle total do servidor;

[2]Common Vulnerability Exposure: https://cve.mitre.org/
[3]Common Vulnerability Exposure Details: http://www.cvedetails.com/
[4]Exploit Database: https://www.exploit-db.com/
[5]National Vulnerability Database: https://nvd.nist.gov/

- Através de inclusão de arquivos e passagem de diretório, verificar permissões de acesso a arquivos de fora do diretório do servidor web pela conta utilizada pelo processo do servidor no sistema;
- Métodos HTTP potencialmente perigosos permitidos pelo servidor web.

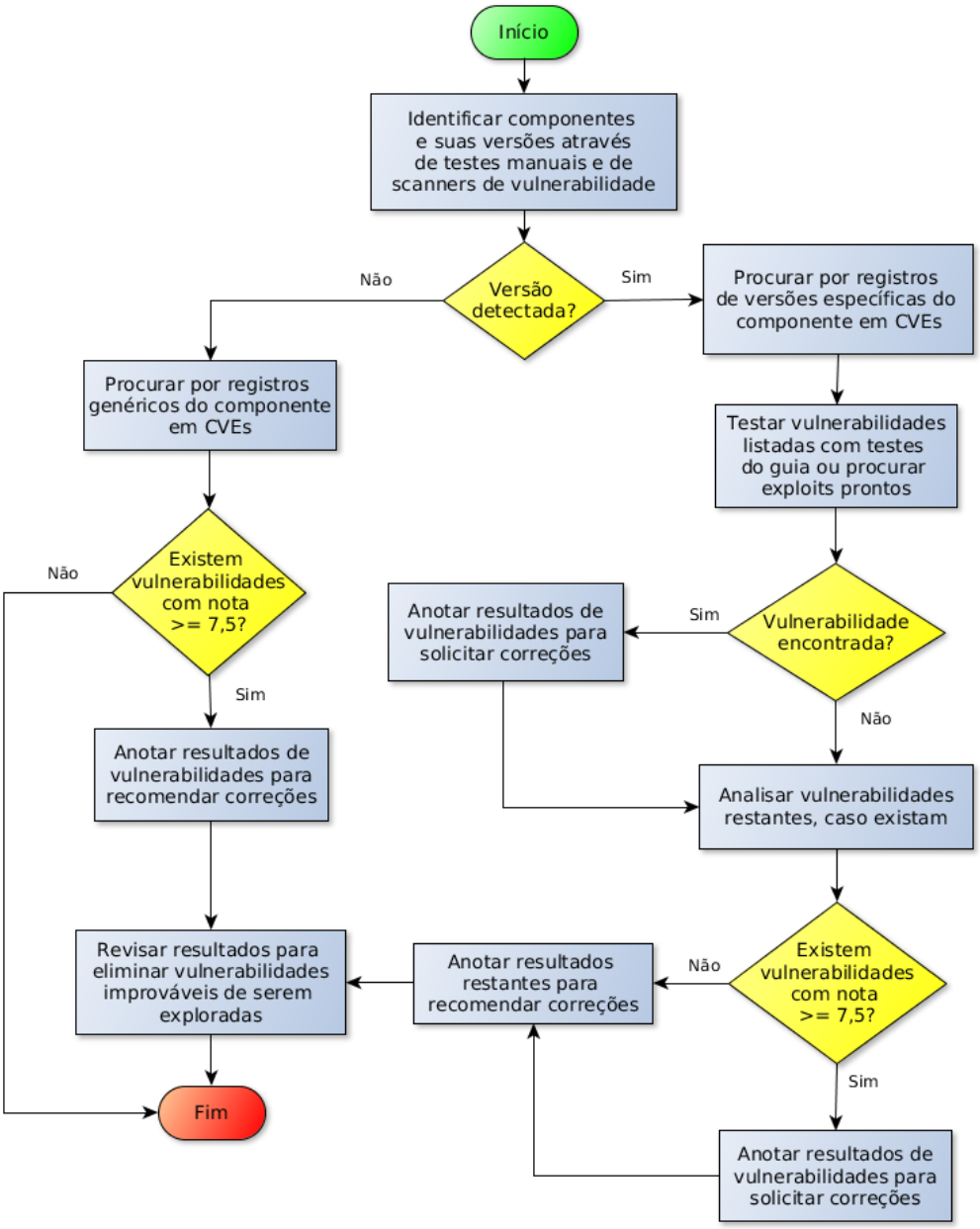
Ao menos as verificações por versões vulneráveis podem ser feitas eficientemente por scanners. Além dos *scanners* de CMS, aplicação e servidor web, são avaliados resultados de *scanners* de servidores, que além de avaliarem componentes relacionados a aplicações web, avaliam qualquer serviço que esteja escutando portas TCP e UDP. Adicionalmente identificam problemas de segurança no próprio sistema operacional. Normalmente são avaliados servidores que hospedam o site alvo e demais componentes subjacentes, como banco de dados.

Os resultados obtidos nas varreduras automatizadas também podem ser uma boa fonte para detectar configurações inseguras. No entanto, para obter eficiência nessa tarefa, é necessário experiência para entender as razões de certas configurações serem inseguras, o que envolve a compreensão do funcionamento de diversas tecnologias e conceitos de segurança da informação.

Esse esforço é levemente reduzido nesse processo justamente pelo mapeamento de tecnologias mais comuns no ambiente de TI da UFRGS, possibilitando que somente as tecnologias mais frequentes sejam estudadas conforme se adquire experiência. *Hackers* éticos precisam entender tecnologias adotadas em cada ambiente testado, para serem efetivos e proverem relatórios de qualidade.

Essas avaliações são consideradas avançadas justamente pela necessidade de haver uma capacidade analítica bem desenvolvida. Analisar resultados de outras vulnerabilidades, ou de varreduras, exigem um conjunto de conhecimentos que não se pode esperar de principiantes. Visto a complexidade dessas análises, foram definidos procedimentos de orientação, que visam avaliar a necessidade de correção das vulnerabilidades. Normalmente, solicitações de correções são feitas caso vulnerabilidades sejam comprovadas ou o CVSS seja alto, conforme o ilustrado na figura 4.6. Essa triagem deve ser feita para evitar repassar mais trabalho aos responsáveis pelo alvo, que frequentemente são relutantes em realizar correções, dispensando vulnerabilidades improváveis ou de severidade baixa.

Figura 4.6: Avaliação de componentes vulneráveis



Fonte: Próprio autor.

4.2.3.4 Registro e atualização de dados

Frequentemente, sites possuem formulários de inscrição em eventos, cujas informações são submetidas aos organizadores por e-mail ou inseridas em um banco de dados, ou então, formulários de cadastro de usuários e atualização de dados cadastrais em um sistema. Diversos sites da UFRGS possuem funcionalidades como essas, que podem estar vulneráveis, e precisam ser testadas. Usualmente, os seguintes procedimentos são realizados:

- Avaliar a necessidade de canais de comunicação criptografados. Dependendo das informações inseridas em um formulário, seja uma inscrição em um evento que exige poucas informações ou registros de usuários em sistemas complexos (com interações entre usuários, por exemplo) podemos concluir se soluções como o uso de HTTPS são realmentes necessárias, ou recomendáveis, e com isso solicitar ou recomendar correções;
- Avaliar necessidade de verificação de ação humana. É muito fácil automatizar um ataque que envie formulários de forma contínua, podendo lotar alguma conta de e-mail ou banco de dados, causando instabilidades em servidores;
- Avaliar se validações de segurança em entradas de usuários são suficientes. Caso seja observado que não há validações em informações submetidas ou que estão fracamente implementadas, é necessário verificar se a falta de sanitização implica em alguma das vulnerabilidades testadas;
- Verificar como as informações inseridas são replicadas no sistema. Exemplificando, se o nome de usuário, que é uma informação que usualmente é replicada diversas vezes em um sistema (perfil de usuário, comentários identificados, entre outros), não possui validação, provavelmente valores maliciosos podem ser inseridos de forma arbitrária constituindo uma vulnerabilidade de severidade grave;
- Verificar possibilidade de enumeração de usuários. Tenta-se criar um usuário já existente, se a aplicação responder que determinado usuário existe e não há validação de ação humana, o site está suscetível a esse problema;
- Avaliar políticas de criação de senha. Caso uma aplicação mais complexa (com mais funcionalidades) aceite senha simples no registro de usuário, é sempre válido recomendar aos responsáveis rever o procedimento.

Além desses problemas terem sido detectados frequentemente em sites mais complexos, não foram encontradas ferramentas que pudessem auxiliar nessas avaliações, exigindo mais experiência dos avaliadores. Por tais motivos, esses testes foram categorizados como avançados.

4.2.3.5 Mecanismos de controle de acesso

Um dos objetivos de se autenticar um usuário no sistema consiste em fornecer meios de negar ou autorizar operações que ele deseja realizar, de acordo com os privilégios que possui, além de permitir o rastreamento do que é feito na aplicação (UTO, 2013).

Para a implementação desses controles, componentes nomeados como monitor de referências (ANDERSON, 1972) devem ser implementados, que para cada ação monitorada, necessita cumprir os seguintes requisitos para ser efetivo:

- Ser inviolável;
- Ser invocado em todo e qualquer acesso a recurso;
- Ser suficiente pequeno para que possa ser analisado e ter sua correção comprovada.

Mecanismos de controle de acesso estão vulneráveis quando permitem que usuários acessem recursos ou funcionalidades das quais não estão autorizados. De acordo com Studdard e Pinto (2011), existem três formas principais de ataque em controles de acesso:

- **Elevação de privilégio horizontal:** Quando, através de falhas, é possível acessar recursos reservados a usuários com mesmo nível de privilégio;
- **Elevação de privilégio vertical:** A mesma situação do item anterior, mas para acesso não autorizado em recursos reservados a usuários de nível de privilégio superior;
- **Exploração de lógica de negócio:** A exploração de lógica de negócios ocorre quando um usuário pode explorar uma falha na máquina de estados de uma aplicação para obter acesso a um recurso-chave. Por exemplo, um usuário pode ser capaz de ignorar a etapa de pagamento em um *check-out* de compras seqüência.

Nos sites da UFRGS, esses problemas foram encontrados exclusivamente em aplicações web mais complexas, identificadas sem o uso de ferramentas. Portanto, os testes para identificação de falhas de controle de acesso foram considerados avançados. Para a identificação dessas falhas, as seguintes testes são frequentemente utilizadas:

- Tentar acessar diretamente funcionalidades reservadas a outros usuários ou níveis de acesso superiores do usuário atual;
- Tentar acessar diretamente arquivos estáticos reservados a outros usuários ou níveis de acesso.
- Procurar por parâmetros de identificação de usuário ou de nível de acesso e alterá-los com o objetivo de obter elevação de privilégios.

4.2.3.6 Cross-Site Request Forgery

A falsificação de solicitações entre sites, ou CSRF, é um ataque no qual comandos não autorizados são transmitidos através de um usuário em quem o site confia. Normal-

mente, o CSRF é utilizado para executar ações de escolha do atacante usando a sessão autenticada da vítima.

Segundo (BLATZ, 2007), transferência de fundos de contas de usuários em aplicações de *internet banking*, roubo de contas a partir de e-mails de usuários para posteriormente utilizar funcionalidade de recuperação de senha, adição de contas de usuários em blogs e verificação de existência de arquivos em intranets são possibilidades de ataques CSRF. Os seguintes fatores são determinantes para uma requisição ser vulnerável a esse tipo de ataque:

- A requisição realiza uma ação privilegiada;
- A aplicação utiliza somente *cookies* HTTP para rastreamento de sessões, não existindo tokens de sessão únicos gerados a cada requisição;
- O atacante consegue determinar todos os parâmetros necessários para realizar a ação. Por exemplo, parâmetros como *"admin"* e *"password"* são facilmente previsíveis em requisições de autenticação. À parte da falta de *tokens* de sessão, não é necessário o uso parâmetros imprevisíveis para combater a vulnerabilidade (STUDDARD; PINTO, 2011).

Nos sites da Universidade, considerando somente sites mais importantes, com funcionalidades sensíveis, potencialmente danosas se exploradas por atacantes, raros casos de CSRF foram identificados. Tendo em vista a dificuldade de induzir responsáveis a providenciar correções, foi concluído não valer a pena o esforço de solicitar correções dessa vulnerabilidade, caso sejam detectadas em sites mais simples e o impacto da exploração for baixo.

Além da baixa frequência de detecção dado as restrições impostas, o entendimento de como explorar CSRF e como corrigir a vulnerabilidade não é trivial, e por isso esses testes foram definidos como avançados. Outro motivo para essa categorização foi a falta de uma ferramenta específica e eficiente para detecção de CSRF, apesar de alguns *scanners* como *Arachni*, apresentarem heurísticas básicas de detecção.

4.3 Apresentação de resultados

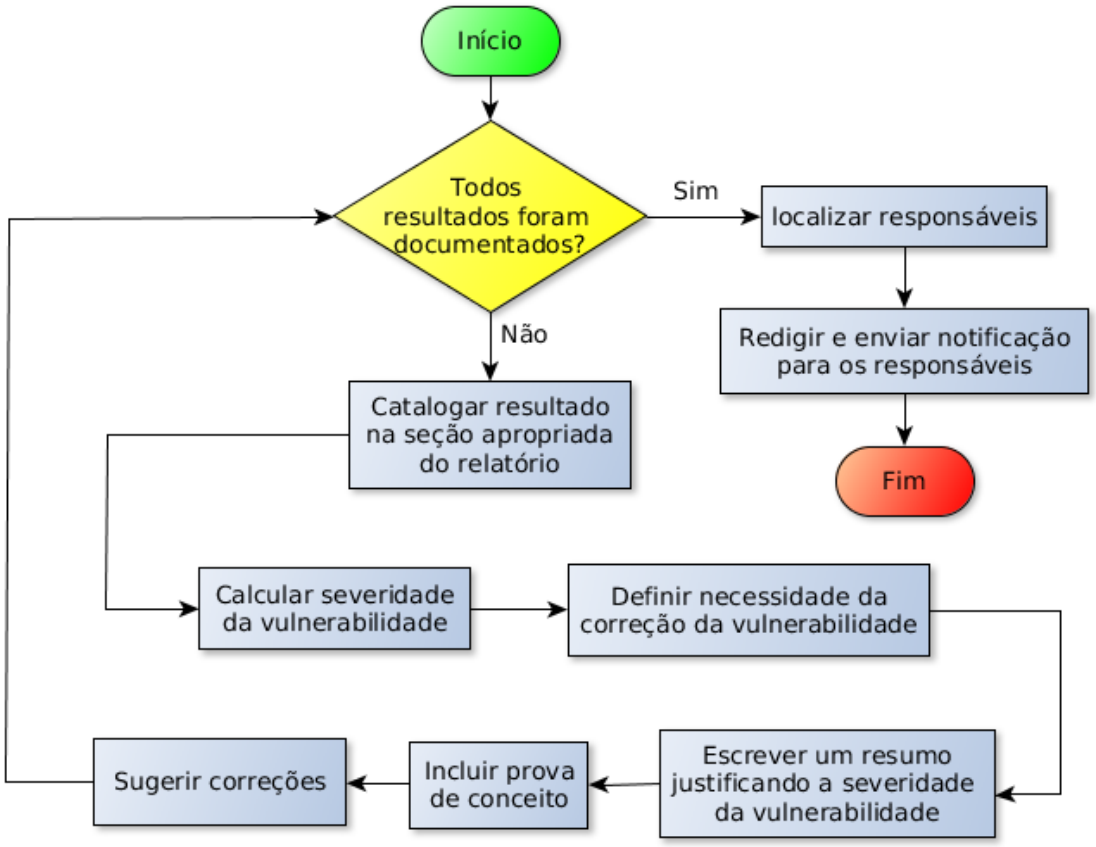
Uma vez finalizado um teste de intrusão, os resultados devem ser apresentados aos responsáveis. Esta etapa, cujos passos estão ilustrados na figura 4.7, é de suma importância, pois é a partir do relatório produzido que as correções poderão ser realizadas.

É essencial que haja preocupação de direcionar as devidas informações para quem puder entendê-las. Correntemente, os responsáveis não possuem conhecimentos técnicos, então muito provavelmente não serão capazes de corrigir as vulnerabilidades reportadas. No entanto, espera-se que tenham contato com os desenvolvedores do site vulnerável.

Para viabilizar essa etapa, decidiu-se notificá-los sobre os problemas de forma a abstrair os detalhes técnicos, visando alertar sobre os impactos das vulnerabilidades, ao invés de informações técnicas, como as classificações das vulnerabilidades ou provas de conceito (demonstrações de exploração das falhas). Adicionalmente, caso informações confidenciais forem obtidas através das vulnerabilidades, inclui-las na notificação mostrou-se eficiente, sensibilizando os envolvidos e engajando-os a tomarem a providências necessárias.

Para toda parte técnica, um documento a parte deve ser redigido, para ser anexado na notificação aos responsáveis. Deve estar explícito na notificação que o relatório técnico está destinado aos desenvolvedores que devem corrigir os problemas.

Figura 4.7: Desenvolvimento e envio do relatório sobre os resultados



Fonte: Próprio autor.

4.3.1 Análise de riscos

Descobrir vulnerabilidades é importante, mas ser capaz de estimar o risco associado ao negócio é tão importante quanto (MEUCCI; MULLER, 2015). A UFRGS, sendo uma instituição pública, normalmente depende dos recursos repassados pelo governo ou de projetos de pesquisa para se manter. Logo, a maioria dos sites existentes não são diretamente necessários a estabilidade financeira da Universidade, diferentemente de *e-commerces*, por exemplo. Salvo exceções, essa observação permitiu criar métricas mais simples para a avaliação dos riscos de se manter as vulnerabilidades encontradas.

Apesar de possíveis falhas em sites dificilmente prejudicarem a situação financeira da Universidade, ao menos de forma direta, ainda assim podem existir problemas que atinjam a imagem da instituição; que prejudiquem serviços dependentes de sistemas de informação *online*; e que exponham informações confidenciais que terceiros confiaram à Universidade. Atrasos na realização de tarefas, ou a possibilidade da Universidade ser processada por informações pessoais expostas, podem ser vistas como prejuízos financeiros.

Tendo em vista os problemas possíveis e a necessidade de um modelo de avaliação de risco simples, devido a rotatividade frequente de bolsistas no DSInf, a estimação do risco associado das vulnerabilidades foi determinada através da definição e associação dos atributos *explorabilidade* e *impacto*. Essas categorizações foram adaptadas da metodologia de estimação de riscos de Meucci e Muller (2015). Para cada vulnerabilidade abordada no manual didático do processo, existem orientações simplificadas de como definir o nível de explorabilidade e impacto de uma vulnerabilidade para então estimar a severidade do risco.

O impacto de uma vulnerabilidade é basicamente o quanto a exploração de uma falha pode ser danosa a um sistema. Para se definir o nível de impacto, normalmente é avaliado o quanto os seguintes fatores, cujas vulnerabilidades em sites da Universidade são capazes de afetar, são prejudicados:

- **Perda de confidencialidade:** Quanta informação é exposta e o quão sensível são essas informações;
- **Perda de integridade:** Quanta informação pode ser corrompida e o quão danificadas podem ficar;
- **Perda de disponibilidade:** Qual a possibilidade de um serviço ser interrompido e o quão vital é para uma organização;

- **Danos à reputação:** Quanto a exploração de uma vulnerabilidade pode prejudicar a reputação de um negócio;
- **Violação de privacidade:** Quanta informação pessoal de terceiros pode ser exposta pela exploração de uma vulnerabilidade.

A partir dessas definições, foram definidos os requisitos que as vulnerabilidades precisam apresentar para definir qual o nível de impacto associado. A ideia por trás dessas definições prévias é auxiliar os iniciantes na avaliação dos resultados dos testes. Normalmente, o impacto de vulnerabilidade pode apresentar os seguintes níveis:

- **Baixo:** Vulnerabilidades que não alteram, ou dificilmente afetam, o estado do sistema (informações armazenadas, disponibilidade do sistema, etc) ou que se restringem a prejudicar visitantes;
- **Médio:** Vulnerabilidades que prejudicam clientes alterando o sistema que acessam, que permitem acessar informações confidenciais pouco sensíveis ou prejudicar a disponibilidade;
- **Grave:** Vulnerabilidades que permitem o atacante realizar ações não permitidas (como acessar informações confidenciais) e modificá- las, basicamente, qualquer falha que resulte em escalada de privilégio.

A explorabilidade é definida por um conjunto de fatores, como a habilidade necessária do atacante, a facilidade de detecção e exploração da vulnerabilidade e o nível de acesso necessário para haver a possibilidade de ataque. A partir desses fatores, os seguintes níveis de explorabilidade foram definidos:

- **Fácil:** Vulnerabilidades que não exigem muita habilidade para serem exploradas, acessíveis e fáceis de descobrir;
- **Médio:** Vulnerabilidades que exigem certa habilidade para serem descobertas ou exploradas e até podem ser facilmente acessíveis;
- **Difícil:** Vulnerabilidades que requerem condições muito restritas para exploração, ou exigir muita habilidade, para ser exploradas somado a necessidade de privilégios de acesso.

Para essa avaliação, a infraestrutura de segurança existente é desconsiderada, portanto a ação de mecanismos de detecção de intrusão não fazem parte da avaliação da explorabilidade. Essa decisão se deve ao fato dos testes serem feitos de dentro da rede, sendo mais difícil avaliar esse requisito, assim como é preferível imaginar que um ata-

cante poderia evadir esses sistemas, expondo o nível real de explorabilidade na aplicação.

Para simplificar a correlação desses atributos, foi definida uma tabela que deve ser consultada após a definição do impacto e explorabilidade de uma vulnerabilidade. Os níveis de cada atributo estão distribuídos na tabela 4.3 de forma a possuírem níveis de severidade da vulnerabilidade para cada combinação possível.

Tabela 4.3: Níveis de severidade de risco em função da explorabilidade e impacto

Severidade do risco		Explorabilidade		
		Difícil	Média	Fácil
Impacto	Grave	Médio	Alto	Crítico
	Médio	Baixo	Médio	Alto
	Baixo	Nulo	Baixo	Médio

Fonte: Próprio autor.

4.3.2 Estruturação dos resultados

Não existe forma obrigatória de estruturar o documento, no entanto, foram definidos procedimentos de organização que podem ser seguidos. Também não há necessidade de seguir todas recomendações em um primeiro momento, esses procedimentos podem ser incluídos em relatórios de forma gradual, conforme se aprende realizar essa atividade, funcionando como uma forma de direcionar o aprendizado. Os seguintes procedimentos podem ser reproduzidos na elaboração de um relatório:

- Utilizar URLs da localização de vulnerabilidades como seções do relatório. Caso mais de uma vulnerabilidade seja descoberta, basta inclui-la na mesma seção;
- Se a vulnerabilidade não puder ser localizada por URLs, como uma configuração insegura em um servidor web, por exemplo, a seção pode ser nomeada com o nome do componente vulnerável;
- Para cada resultado, estimar a severidade do risco e escrever um resumo versando sobre a possibilidade de detecção, nível técnico necessário do atacante e o impacto da vulnerabilidade para os responsáveis e terceiros;
- Se possível, incluir também uma prova de conceito possibilitando a reprodução do ataque, e sugerir possíveis correções para as vulnerabilidades.

4.4 Monitoramento de correções

Infelizmente, como raramente esses testes são solicitados, certo esforço é necessário para que se garanta que as correções sejam realizadas. Conforme já discutido neste trabalho, profissionais de *pentest* são contratados para realizar um serviço, que é finalizado assim que um relatório com os resultados encontrados é entregue aos contratantes. Consequentemente, nenhum material consultado apontou soluções para esse problema, exigindo a criação de uma etapa específica para a acompanhar as correções das vulnerabilidades.

O objetivo dessa etapa é, de forma organizada, conseguir obter ao menos uma resposta dos responsáveis, para garantir que estão cientes das vulnerabilidades e também pressioná-los a corrigi-las, tomando medidas punitivas, em caso de negligência. Basicamente, após obtidos os resultados, é necessário reportá-los aos responsáveis. Após enviada a notificação, é recomendado esperar até três dias por respostas caso vulnerabilidades de severidade grave tenham sido encontradas, e, até sete dias caso contrário. Esses períodos de espera foram definidos a partir da observação dos tempos de respostas para incidentes de vulnerabilidades em sites detectados antes da concepção deste trabalho.

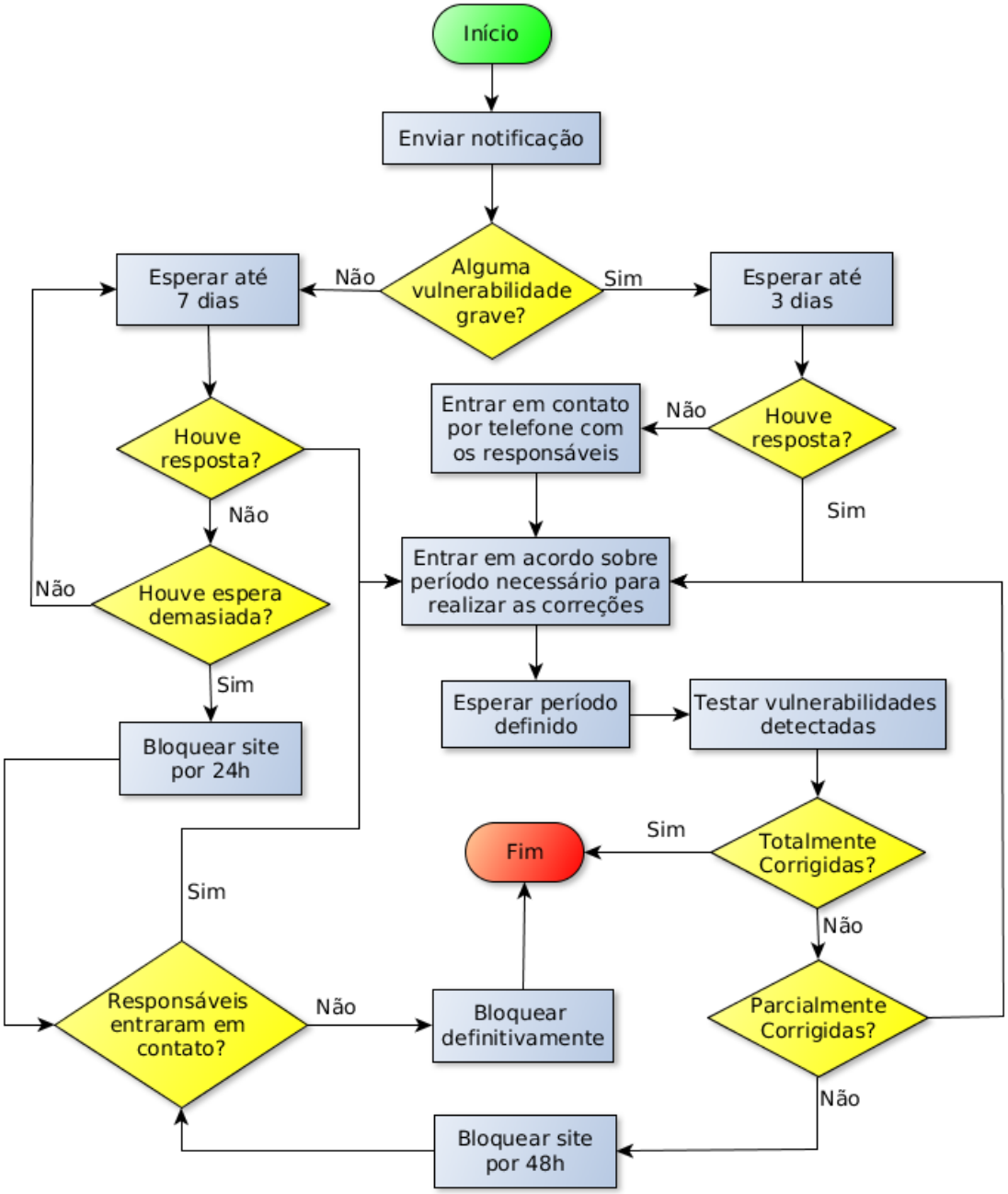
No primeiro caso, devido a gravidade do caso, é recomendado entrar em contato por telefone com os responsáveis. A partir desse contato, é definido quanto tempo será necessário para as correções ou para tirar o site do ar, caso seja antigo e abandonado e todas as partes envolvidas concordem. Se as correções não foram totalmente efetivas, é necessários reportar as vulnerabilidades que ainda existem repetindo o ciclo.

No segundo caso, pode haver mais tolerância e o período de espera é sete dias para enviar uma nova notificação. Se por acaso muitas notificações forem enviadas, sem que hajam respostas, é permitido agir de forma mais proativa, como forma de alerta.

Independente da situação, na ocorrência das notificações serem totalmente negligenciadas, ou o período estipulado para as correções não seja cumprido, é possível bloquear o site por um curto período (como dois dias). Caso ninguém tenha sentido a falta do site, muito provavelmente está abandonado, possibilitando o bloqueio permanente.

Por motivos de simplicidade, o fluxograma da figura 4.8 não considera a importância dos sites para a Universidade. Por isso, o uso do bom senso é importante para essa etapa, alguns sites simplesmente não podem ser removidos e, na ocorrência de não serem corrigidos, pode ser necessário relevar as vulnerabilidades e investigar estratégias para mitigação de riscos.

Figura 4.8: Processo de acompanhamento de correções



Fonte: Próprio autor.

5 AVALIAÇÃO EXPERIMENTAL

Nesta seção pretende-se expor a análise dos resultados da aplicação da estratégia pelo autor e terceiros. A intenção é visualizar a efetividade da organização dos testes observando os resultados encontrados por bolsistas de diferentes níveis de experiência e do próprio autor.

5.1 Metodologia

Para atingir o objetivo de avaliar a efetividade da estratégia desenvolvido neste trabalho, são coletados resultados de testes feitos pelo autor, com maior experiência, e de outro integrante da equipe, com pouca experiência e pouco tempo atuando no TRI, em ambientes de diferentes níveis de complexidade. Adicionalmente, é coletada a quantidade de informações definidas como importantes pela estratégia que cada testador conseguiu obter, assim como o tempo necessário para avaliar os ambientes de teste.

Os dados obtidos e os critérios observados são utilizados para uma análise da eficácia da organização da estratégia. Esta seção fornece detalhe dos dois participantes da avaliação, dos cenários de teste e dos critérios de comparação utilizados.

5.1.1 Critérios para análise

A eficiência da estratégia é medida através da capacidade de atender os requisitos definidos dado os problemas enfrentados pelo DSInf, como o fato da tarefa de avaliar aplicações web serem delegadas a bolsistas que por sua vez possuem alta rotatividade na equipe. Assim, correlaciona-se o nível de experiência dos participantes da avaliação com a complexidade dos ambientes testados e a distribuição das vulnerabilidades encontradas por cada participante dado a complexidade dos testes utilizados. Adicionalmente, avalia-se o tempo necessário pelos participantes para testar cada alvo, assim como a quantidade de informação obtida na fase de reconhecimento.

Para um conjunto de testes, são variados o nível de experiência do testador e a complexidade do alvo testado, conforme a definição de complexidade de ambientes presente neste trabalho. Desses conjuntos, são avaliados os seguintes pontos:

- A quantidade de resultados oriundos dos testes básicos e avançados por cada participante. Essa informação permite avaliar se os testes foram alocados corretamente em cada categoria, com o domínio dos testes acompanhando a evolução da experiência dos testadores;
- A efetividade da cobertura de vulnerabilidades descobertas pelos testes básicos, com o objetivo de verificar se é possível haver bolsistas aptos a realizarem testes eficientes em pouco tempo de aprendizado, sem o total domínio da estratégia;
- O tempo necessário para realizar os testes em cada ambiente pelos participantes, o que permite avaliar a influência da experiência no tempo necessário para testar um alvo, sem deixar de considerar a quantidade de vulnerabilidades encontradas. Dada a quantidade de sites da UFRGS que precisam ser avaliados, pode ser interessante visualizar reduções no tempo médio necessário para testar alvos ou no aumento da média da quantidade de vulnerabilidades encontradas pelo tempo dispensado, evidenciando o aumento no rendimento dos bolsistas conforme se domina a estratégia;
- A capacidade dos participantes reconhecerem informações de um ambiente. É importante avaliar se a estratégia é bem sucedida conscientizar das vantagens da correta realização da fase de reconhecimento, assim como instruir os bolsistas a reconhecerem ambientes mais críticos, que exigem testes mais cuidadosos.

5.1.2 Participantes

Os participantes da avaliação foram exclusivamente bolsistas do DSInf. Uma breve comparação das características dos participantes está presente na tabela 5.1.

Tabela 5.1: Comparação entre os participantes da avaliação

Participante	Bolsista A (Autor)	Bolsista B
Curso	Engenharia de computação	Ciência da computação
Semestre	10º	4º
Experiência	2 anos e 2 meses	1 mês e 2 semanas

Fonte: Próprio autor.

Infelizmente, durante o desenvolvimento deste trabalho, um dos bolsistas saiu da equipe, restando somente o autor da estratégia e outro bolsista menos experiente. Ainda

assim, devido a diferença de experiência, foi possível tentar visualizar se o trabalho desenvolvido consegue satisfazer o requisito de ser útil independente da habilidade de quem realiza os testes.

A ideia inicial era poder contar com bolsistas de outros departamentos e funcionários do DSInf, para se obter mais dados para se analisar e dar maior credibilidade para os experimentos. No entanto, o primeiro grupo está indisponível justamente por estarem envolvidos com as atividades dos seus respectivos departamentos no CPD, enquanto o segundo grupo, que possui diversas responsabilidades, sofreu com a saída de funcionários durante o desenvolvimento deste trabalho, sendo inviável a reserva de tempo para as avaliações experimentais da estratégia.

5.1.3 Cenários para teste

Dois alvos distintos, com diferentes funcionalidades e importância para a Universidade foram escolhidos para a avaliação. A escolha dos ambientes levou em consideração a complexidade do desenvolvimento da aplicação, a quantidade e importância das informações manipuladas e a frequência de funcionalidades potencialmente inseguras.

O primeiro ambiente consiste em um sistema de pesquisa de trabalhos desenvolvidos na área de conhecimento de uma das unidades acadêmicas da UFRGS. A aplicação possui um banco de dados que realiza somente leituras e está hospedada em um ambiente compartilhado, necessitando certo cuidado do bolsista ao realizar os testes.

O segundo ambiente é uma plataforma de ensino a distância desenvolvida na Universidade, com registro de usuários, níveis de privilégio, interações entre usuários através de mensagens privadas e salas de conversas para as turmas registradas no sistema. O sistema possui um banco de dados com diversas informações sigilosas, e é possível inserir e alterar dados em diversas operações disponibilizadas, além de também possuir funcionalidades que interagem com servidores de email. Esse serviço é considerado crítico para a Universidade, portanto também exige cuidado do bolsista ao testá-lo

Apesar da diferença de complexidade dos dois sistemas, ambos exigem cautela ao serem avaliados. A decisão de exigir prudência dos participantes dos experimentos nos dois casos advém da real necessidade da noção de responsabilidade ao realizar testes de intrusão independente do nível de experiência.

5.2 Análise dos resultados

O primeiro resultado avaliado foi a quantidade de informação obtida dos ambientes pelos testadores, disponível na tabela 5.2. Independente do nível de experiência dos participantes e da complexidade do alvo, durante a fase de reconhecimento e de testes de vulnerabilidade, ambos participantes foram capazes de obter informações reconhecidas como importantes.

Tabela 5.2: Quantidade de informação obtida pelos participantes em cada ambiente

Experiência do participante	Complexidade do alvo	Linguagem	SGBD	Servidor web	SO	Ambiente especial
Iniciante	Simples	✓	✓	✓	✓	✓
	Avançado	✓	✓	✓	✓	✓
Avançado	Simples	✓	✓	✓	✓	✓
	Avançado	✓	✓	✓	✓	✓

Fonte: Próprio autor.

A única informação de reconhecimento obrigatório é a identificação de alvos presentes no ambiente de hospedagem do CPD, ou de serviços críticos. Em relação a essa necessidade, as orientações da estrategia de testes foram bem sucedidas.

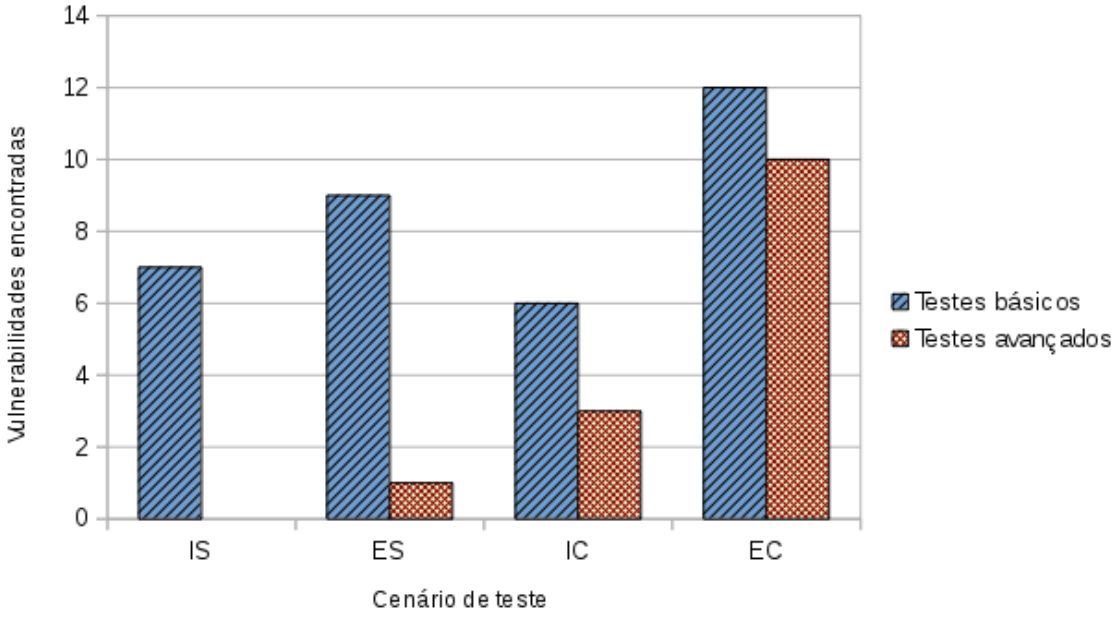
A figura 5.1 apresenta a quantidade de vulnerabilidades encontradas dado a categoria do teste utilizada por cada participante nos dois ambientes avaliados. Para simplificar o gráfico, o eixo horizontal está dividido por cenários de teste que combinam a experiência do participante e a complexidade dos ambientes, definidos na tabela 5.3.

Tabela 5.3: Cenários de teste definidos por categoria

Cenário de teste	IS	ES	IC	EC
Participante	Iniciante	Experiente	Iniciante	Experiente
Ambiente	Simples	Simples	Complexo	Complexo

Fonte: Próprio autor.

Figura 5.1: Vulnerabilidades encontradas por categoria de teste



Fonte: Próprio autor.

Em todos os casos, vulnerabilidades descobertas a partir de testes básicos foram encontradas. Esses resultados revelam certo sucesso da estratégia de testes em instruir novos aprendizes a avaliarem aplicações web.

Apesar da diferença da quantidade de vulnerabilidades encontradas, é necessário levar em consideração que o participante menos experiente recebeu somente 6 semanas de treinamento. Esses resultados já seriam considerados suficientes para aplicar testes de intrusão de forma regular nos sites da UFRGS.

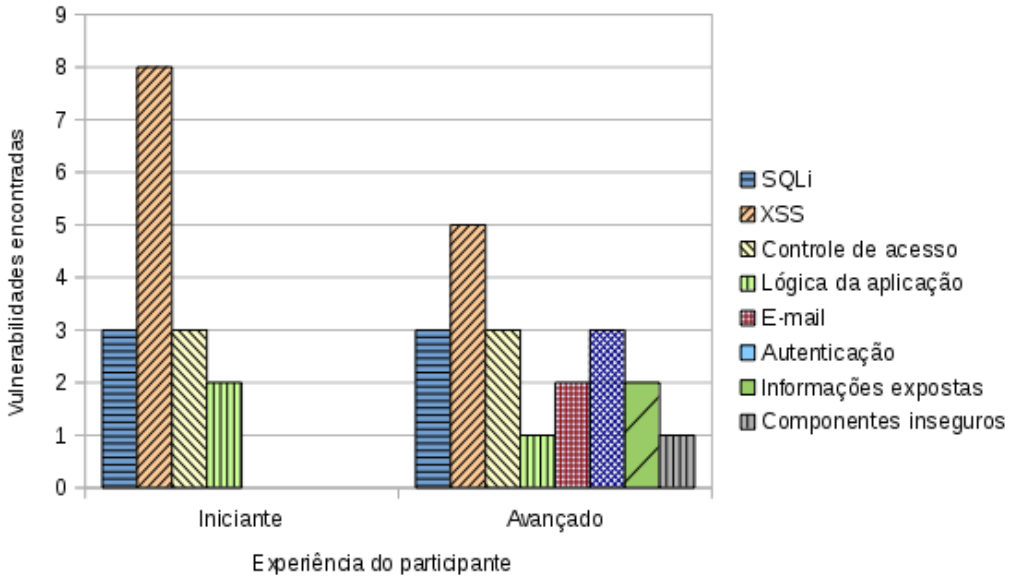
Em ambos os alvos, vulnerabilidades também foram detectadas a partir de testes avançados. O participante com menos experiência recebeu treinamento para esses testes, mas não foi cobrado que obtivesse resultados derivados desses testes na avaliação da estratégia.

O ambiente mais simples apresenta uma vulnerabilidade relacionada à interação com sistemas de e-mail, que foi detectada somente pelo autor. No entanto, o ambiente mais complexo apresentava falhas em mecanismos de controle de acesso que foram descobertas pelo aprendiz. Isso pode ser visto como um bônus, pois não foi cobrado o domínio de nenhum teste da categoria avançada pelo aprendiz.

Considerar a realocação dos testes dessa vulnerabilidade para a categoria de testes básicos exigiria a participação de mais bolsistas de experiência equivalente. Também seria necessário que mais sites fossem testados para observar se ao menos a maioria dos participantes apresentassem o domínio da técnica necessária.

De maneira mais detalhada, na figura 5.2 é apresentada as vulnerabilidades, incluindo o total de incidências nos ambientes avaliados, que foram encontradas pelos participantes. Conforme o esperado, a cobertura de vulnerabilidades detectadas pelo autor da estratégia foi maior, no entanto, o participante menos experiente obteve os mesmos resultados para injeções SQL e falhas em controle de acesso, além de, surpreendentemente, ter detectado mais vulnerabilidades XSS.

Figura 5.2: Vulnerabilidades encontradas por cenário de teste



Fonte: Próprio autor.

Conforme os resultados do capítulo 3, injeções SQL e XSS foram as vulnerabilidades mais frequentes nos sites testados. Portanto, garantir que aplicantes da estratégia aprendam a detectar essas vulnerabilidades em pouco tempo satisfaz a necessidade de se obter testadores relativamente eficientes, sendo capazes de inicialmente detectar vulnerabilidades mais frequentes para com o tempo estender o conjunto de vulnerabilidades que estão aptos a detectar.

Por último, foi avaliado o tempo necessário dos participantes para executarem um teste de segurança nos ambientes eleitos. Também foi observardo o rendimento considerando quantas vulnerabilidades foram detectadas por unidade de tempo.

A tabela 5.4 apresenta as informações necessárias para avaliação da influência da experiência no rendimento do testador. O ambiente de teste mais complexo, presente nos cenários IC e EC, demandou mais tempo para ambos participantes justamente por prover mais funcionalidades ao usuário. Consequentemente, foi necessário avaliar mais páginas e pontos de entrada de informação, o que demandou mais tempo.

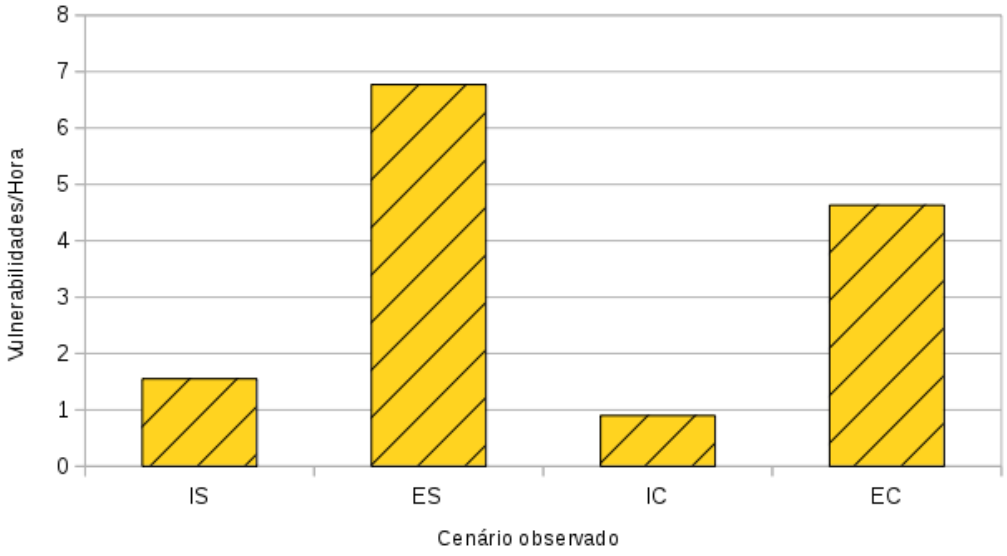
Tabela 5.4: Cenários de teste e tempos aproximados de execução

Cenário observado	IS	ES	IC	EC
Tempo necessário	4h30m	1h20m	10h	4h45m
Vulnerabilidades	7	9	9	22

Fonte: Próprio autor.

A partir do gráfico exibido na figura 5.3, que apresenta a média de vulnerabilidades encontradas por hora, em cada cenário de teste, é possível extrair informações em relação a diferença de rendimento dos participantes. O autor da estratégia, testador nos cenários ES e EC, obteve um rendimento médio aproximadamente 465% superior ao participante menos experiente, testador nos cenários IS e IC.

Figura 5.3: Média de vulnerabilidades encontradas por hora em cada cenário



Fonte: Próprio autor.

Essa constatação permite realizar algumas projeções. Conforme se ganha experiência atuando no DSInf, exercendo diversas atividades que requeiram o aprendizado de conhecimentos técnicos e a aplicação de conceitos de segurança da informação, muito provavelmente reflitam na capacidade de avaliar a segurança de aplicações web mais rapidamente.

O que é conveniente, devido ao fato de haver um número considerável de sites que necessitam ser avaliados. Sendo um dos motivos que justificaram os esforços para criação de uma estratégia específica para o CPD.

5.3 Considerações finais

Basicamente, foram obtidos os resultados esperados. A experiência influe tanto na quantidade de resultados obtidos, quanto no tempo necessário para avaliar aplicações. A estratégia também é capaz de acelerar o aprendizado de bolsistas dispostos a realizar testes de intrusão, possibilitando que sejam relativamente eficientes em um espaço curto de tempo. Ou seja, a estratégia consegue ser satisfatória para diferentes níveis de experiência, além de se apresentar como uma possível solução aos problemas apresentados na subseção 3.2.1.

Apesar do sucesso relativo alcançado, houveram problemas de tempo e falta de pessoas para uma avaliação mais confiável deste trabalho. Devido ao tempo livre disponibilizado para essa atividade no CPD, pois ambos participantes constantemente desviaram o foco para outras demandas do DSInf, impossibilitaram a avaliação de mais alvos de diferentes níveis de complexidade. A adição de sites possibilitaria a visualização de padrões de resultados em aplicações de complexidade aproximada. Também seria possível avaliar a eficiência de participantes em aplicações com funcionalidades distintas, consequentemente com a distribuição de vulnerabilidades diferenciadas, incrementando a avaliação da eficácia dos testes previstos pela estratégia.

No início deste trabalho, além do autor e o bolsista iniciante, havia outro bolsista com bastante experiência na equipe. No entanto, durante o desenvolvimento do trabalho, ocorreu sua saída do CPD antes do tempo previsto, o que prejudicou a qualidade da avaliação. A inclusão de um participante mais experiente permitiria avaliar, com um grau maior de confiança, se haveriam diferenças na aplicação da estratégia por pessoas mais experientes na execução de atividades de segurança da informação. Além disso, se houvessem mais participantes, seria possível visualizar padrões de resultados entre pessoas com experiência aproximadas. O que poderia dar maior credibilidade às conclusões realizadas.

Entretanto, os resultados obtidos não podem ser ignorados, pois apresentaram benefícios reais devido a capacitação de um integrante da equipe em realizar testes de intrusão, que em 6 semanas de treinamento conseguiu detectar aproximadamente 51,2% da quantidade de vulnerabilidades detectadas pelo autor (informação derivada da figura 5.1). Portanto, o desenvolvimento da estratégia deve ser continuado e repassado a futuros integrantes do DSInf, com o objetivo de melhorar e perpetuar esse serviço no CPD.

6 CONCLUSÕES

No decorrer deste trabalho, foram contextualizados os conceitos e tecnologias referentes a aplicações web e avaliações de segurança, principalmente testes de intrusão. Inicialmente foram listadas as principais técnicas utilizadas para avaliar a segurança de aplicações web, com o objetivo de posteriormente, no decorrer do trabalho, justificar os motivos de somente testes de intrusão estarem disponíveis ao DSInf. Tais motivos se sustentam devido ao fato da maioria das aplicações presentes nos servidores da Universidade já estarem em fases tardias de um ciclo de desenvolvimento de software, onde comumentemente essa técnica é aplicada.

O objetivo deste trabalho foi apresentar uma estratégia de avaliação de segurança de sites da UFRGS pelo CPD, sendo essa tarefa delegada internamente ao DSInf. Além de ter sido justificada a escolha da técnica de teste de intrusão como base dessa estratégia, foram apresentados os motivos da necessidade de elaboração de uma solução específica para o ambiente da Universidade, evidenciando as diferenças em relação a empresas e até outras universidades.

A estratégia organizou os testes em quatro etapas. Na primeira etapa, foram definidas formas de se obter alvos, que compreendiam desde a procura por alertas de ataques em ferramentas de segurança a buscas apropriadamente parametrizadas em mecanismos de busca públicos. Formas automatizadas para o mapeamento, assim como informações importantes priorizadas no reconhecimento desses alvos foram apresentadas. Essas informações foram divididas em duas categorias, para fins de priorização de esforços de acordo com a habilidade do aplicante.

Posteriormente, para segunda etapa, foram apresentadas *scanners* de vulnerabilidades, que estavam de acordo com requisitos de adoção apresentados durante o trabalho, para aplicações, servidores web, CMS e servidores hospedeiros. Também foram apresentados testes para diversas vulnerabilidades. Esses testes foram divididos em duas categorias de complexidade. Uma categoria para testes de fácil execução, ou amparados por ferramentas, com o objetivo de serem aprendidos e aplicados por pessoas com pouca experiência em pouco tempo. A segunda categoria reúne testes de maior complexidade de compreensão e execução, para serem aprendidos conforme se ganha experiência realizando testes que estejam de acordo com a estratégia desenvolvida.

As duas últimas etapas foram concebidas para a apresentação de resultados aos responsáveis, além do monitoramento das correções. Foram definidos passos para desen-

volver relatórios técnicos de forma facilitada, para os responsáveis pela correção, assim como relatórios direcionados aos responsáveis administrativos pelo site, muitas vezes sem conhecimentos técnicos. Devido ao histórico de negligência, ou resistência, desses responsáveis, a última etapa foi inteiramente destinada a defrontar com essa situação, a partir de contatos constantes e medidas punitivas.

A concepção da estratégia foi baseada, principalmente, sob o argumento de auxiliar os testadores a se desenvolverem e serem relativamente eficientes mesmo com pouca experiência. O que constitui em uma forma de resolver o problema da rotatividade de bolsistas, cuja tarefa é delegada, no DSInf. Para avaliar a efetividade dessas medidas, um experimento com participantes com diferentes níveis de experiência e alvos de complexidade distintas foi realizado. Os resultados da avaliação possibilitaram diversas constatações positivas sobre a efetividade da estratégia. No entanto, infelizmente, essa avaliação apresentou diversas deficiências.

Primeiro, o pouco tempo disponibilizado para a avaliação de sites no CPD impossibilitou que mais alvos fossem testados, pois foi necessário dedicar tempo a outras demandas do DSInf. Consequentemente, a confiabilidade da estruturação da estratégia foi reduzida, pois se baseou na observação de padrões em sites da Universidade, além dos resultados dos testes. O segundo problema foi a falta de pessoas disponíveis para participar da avaliação, o que prejudicou a análise da influência da experiência do testador nos resultados da aplicação da estratégia. Por último, a qualidade dos relatórios e eficiência do monitoramento de correções não foram avaliados. Devidos as outras atividades do DSInf, não foi possível destinar tempo suficiente para planejar e gerênciar notificações para todos 50 sites testados.

Ainda assim, o desenvolvimento deste trabalho apresentou benefícios reais. A capacitação de um integrante da equipe, assim como a detecção de diversos sites vulneráveis, que estão em processo de correção, o que ocasiona no aumento da segurança da rede da Universidade, foram as contribuições deste trabalho.

Como trabalhos futuros, existe a necessidade de continuar a desenvolver a estratégia. Em relação ao total de sites existentes na Universidade, a proporção avaliada é pequena, sendo possível a modificação do trabalho realizado a partir de novas estatísticas provenientes de um número maior de sites avaliados. Outro motivação para o desevolvimento contínuo da estratégia é a constante adoção de novas tecnologias em ambientes web. Nos sites avaliados não foi encontrada a presença de banco de dados NoSQL, algo que atualmente é constante em aplicações desenvolvidas em empresas, portanto injeções

para essa espécie de banco de dados não foram consideradas na estratégia, mas é algo que pode mudar com o passar dos anos.

Outra abordagem poderia ser a expansão da cobertura da estratégia para outras técnicas de avaliação de segurança em aplicações web considerando futuras possíveis melhorias na organização do ambiente de TI, da PSI da UFRGS, ou ainda, para outras áreas como testes de intrusão em infraestrutura e redes sem fio. Por último, para um melhor aproveitamento em um contexto acadêmico, não visando somente o ambiente administrado pelo CPD. Este trabalho poderia ser uma base para o desenvolvimento de uma metodologia de avaliação de segurança de aplicações web caixa-branca. Nesse caso, o trabalho seria direcionado a profissionais da área de segurança da informação, que se apoiariam em um material que auxiliaria na criação de uma estratégia específica para empresas, ou instituições públicas, das quais fazem parte.

REFERÊNCIAS

ACUTINEX. **Acutinex Web Vulnerability Scanner**. 2017. Disponível em: <https://www.acunetix.com/web-vulnerability-manager/>. Acesso em: 16 abr. 2017.

ALCORN, W. **BeEF - The Browser Exploitation Framework Project**. 2017. Disponível em: <http://beefproject.com/>. Acesso em: 15 abr. 2017.

ANDERSON, J. P. **Computer Security Technology Planning Study**. [S.l.], 1972. v. 2.

AUSTIN A. WILLIANS, L. One technique is enough: A comparison of vulnerability discovery techniques. In: **2011 5th International Symposium on Empirical Software Engineering and Measurement**. Railegh, USA: IEEE, 2011. p. 91–106.

BACUDIO, A. G. et al. An overview about penetration testing. **International Journal Of Network Security and Its Applications**, v. 3, n. 6, p. 19–38, nov. 2011.

BALLAD, T.; BALLAD, W. **Securing PHP Web Applications**. [S.l.]: Addison-Wesley, 2009.

Berkeley Information Security and Policy. **Database Hardening Best Practices**. 2017. Disponível em: <https://security.berkeley.edu/resources/best-practices-how-articles/database-hardening-best-practices>. Acesso em: 9 mai. 2017.

BIRD, T. **Choosing Between a Penetration Test and a Secure Code Review**. 2016. Disponível em: <https://dzone.com/articles/choosing-between-penetration>. Acesso em: 14 abr. 2017.

BLATZ, J. **CSRF: Attack and Defense**. Santa Clara, US, 2007.

BOYD, S. W.; KEROMYTIS, A. D. Sqlrand: Preventing sql injection attack. In: ACNS, 2004, Yellow Mountain, CN. **2nd Applied Cryptography and Network Security Conference**. New York, US: Springer, 2004. p. 293–302.

CENTRO DE ESTUDOS, TRATAMENTO E RESPOSTA DE INCIDENTES DE SEGURANÇA NO BRASIL. **Cartilha de Segurança para Internet**. 2012. Disponível em: <http://cartilha.cert.br/>. Acesso em: 14 abr. 2017.

CERON, J. et al. Sistema de registro de estações da ufrgs como ferramenta de segurança. **IV Workshop de Tecnologia da informação das IFES**, Rio de Janeiro, Brasil, 2010.

CONKLIN, L.; ROBINSON, G. **Code Review Guide 2.0 Alpha Release**. 2nd. ed. [S.l.]: The OWASP Foundation, 2016.

CONSELHO UNIVERSITÁRIO DA UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL. **Política de Segurança da Informação da UFRGS**. 2014. Disponível em: <http://www.ufrgs.br/consun/legislacao/documentos/Dec124-14%20-%20Politica%20de%20Seguranca%20da%20Informacao%20da%20UFRGS.pdf>. Acesso em: 14 abr. 2017.

CROSS, M. et al. **Web Application Vulnerabilities: Detect, Exploit, Prevent**. Burlington, USA: Syngress, 2007.

DAMELE, B.; STAMPAR, M. **sqlmap: automatic SQL injection and database takeover tool**. 2017. Disponível em: <http://sqlmap.org>. Acesso em: 15 abr. 2017.

DOUPÉ, A.; COVA, M.; VIGNA, G. Why johnny can't pentest: An analysis of black-box web vulnerability scanners. In: DIMVA, 2010, Bonn, GER. **DETECTION OF INTRUSIONS AND MALWARE, AND VULNERABILITY ASSESSMENT: 7TH INTERNATIONAL CONFERENCE**. Berlin, GER: Springer, 2010. p. 111–131.

DVWA Team. **DVWA - Damn Vulnerable Web Application**. 2017. Disponível em: <http://www.dvwa.co.uk/>. Acesso em: 23 abr. 2017.

ECMA International. **ECMAScript language specification. Standard ECMA-262**. 2016. Disponível em: <https://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf>. Acesso em: 15 abr. 2017.

Edge-security Group. **Paros Proxy**. 2013. Disponível em: <http://www.edge-security.com/proxystrike.php>. Acesso em: 16 abr. 2017.

FIORIO, M. et al. Soluções para o desenvolvimento de sistemas seguros. **VII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais**, Rio de Janeiro, Brasil, p. 153–193, 2007.

Forum of Incident Response and Security Teams. **Common Vulnerability Scoring System**. 2015. Disponível em: <https://www.first.org/cvss>. Acesso em: 9 mai. 2017.

FOUNDATION, T. O. **OWASP Zed Attack Proxy**. 2017. Disponível em: <https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project>. Acesso em: 16 abr. 2017.

GREENBONE. **OpenVAS - Open Vulnerability Assessment System**. 2017. Disponível em: <http://www.openvas.org/>. Acesso em: 16 abr. 2017.

GROSSMAN, J. et al. **XSS Attacks: Cross Site Scripting Exploits and Defense 1st Edition**. 1st. ed. Burlington, US: Syngress, 2007.

GUTZMANN, K. Access control and session management in http environment. **IEEE Internet Computing**, IEEE, New York, USA, v. 5, n. 1, p. 26–35, Aug 2001.

HALFOND, W. G. J.; VIEGAS, J.; ORSO, A. A classification of sql injection attacks and countermeasures. In: **Proceedings of the 28th international conference on Software engineering**. New York, US: ACM, 2006.

HERZOG, P. **OSSTMM 3 - The Open Source Security Testing Methodology Manual**. 3rd. ed. [S.l.]: ISECOM, 2010.

HOEPMAN, J.; JACOBS, B. Increased security through open source. **Communications of the ACM**, New York, USA, v. 50, n. 1, p. 79–83, jan. 2007.

HUBCZYK, M.; DOMANSKI, A.; DOMANSKA, J. Local and remote file inclusion. In: EWARYST, T.; KAPCZYNSKI, A. (Ed.). **Internet - Technical Development and Applications**. Berlin, DE: Springer, 2012. chp. 17, p. 189–200.

JERKOVIć, H.; VRANEšIć, P.; DADIć, S. Securing web content and services in open source content management systems. In: MIPRO, 2016, Opatija, HR. **39th International Convention on Information and Communication Technology, Electronics and Microelectronics**. Zagreb, HR: IEEE, 2016.

KARIM, I. **Fimap**. 2015. Disponível em: <https://tha-imax.de/git/root/fimap/tree/master>. Acesso em: 16 abr. 2017.

LANCOR, L.; WORKMAN, R. Using google hacking to enhance defense strategies. In: **SIGCSE '07 Proceedings of the 38th SIGCSE technical symposium on Computer science education**. New York, USA: ACM, 2007. v. 39, n. 1, p. 491–495.

LASKOS, A. **Arachni - Web Application Security Scanner Framework**. 2017. Disponível em: <http://www.arachni-scanner.com/>. Acesso em: 16 abr. 2017.

LAWTON, G. Web 2.0 creates security challenges. **Computer**, California, USA, v. 40, n. 10, p. 13–16, oct. 2007.

LYON, G. **Nmap: the Network Mapper**. 2016. Disponível em: <https://nmap.org/>. Acesso em: 16 abr. 2017.

MCGRAW, G. **Software Security: Building Security**. 1st. ed. [S.l.]: Addison-Wesley, 2006.

MEUCCI, M.; MULLER, A. **OWASP Testing Guide 4.0**. 4th. ed. [S.l.]: The OWASP Foundation, 2015.

MUNIZ, J.; LAKHAMI, A. **Web Penetration Testing with Kali Linux**. Birmingham, UK: Packt Publishing, 2013.

NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY. **Risk management guide for information technology systems**. 2012. Disponível em: <http://csrc.nist.gov/publications/nistpubs/800-30-rev1/sp800_30_r1.pd>. Acesso em: 15 abr. 2017.

NIC BR Security Office. **Práticas de Segurança para Administradores de Redes Internet**. 2003. Disponível em: <https://www.cert.br/docs/seg-adm-redes/seg-adm-redes.pdf>. Acesso em: 15 abr. 2017.

PAULI, J. **The Basics of Web Hacking: Tools and Techniques to Attack the Web**. 1st. ed. Waltham, USA: Elsevier Science, 2013.

PENETRATION TESTING EXECUTION STANDARD. **Penetration Testing Execution Standard**. 2014. Disponível em: <http://www.pentest-standard.org/index.php/Main_Page>. Acesso em: 15 abr. 2017.

PortSwigger Web Security. **Burp Suite**. 2017. Disponível em: <https://portswigger.net/burp/>. Acesso em: 16 abr. 2017.

RAPID7. **Metasploit: Penetration Testing Software**. 2017. Disponível em: <https://www.metasploit.com/>. Acesso em: 16 abr. 2017.

RIANCHO, W. **W3af - Web Application Attack and Audit Framework**. 2017. Disponível em: <http://www.w3af.org/>. Acesso em: 16 abr. 2017.

SCRT Information Security. **Webshag**. 2009. Disponível em: <https://www.scrt.ch/en/attack/downloads/webshag>. Acesso em: 16 abr. 2017.

SHKLAR, L.; ROSEN, R. **Web application architecture: principles, protocols, and practices**. 1st. ed. [S.l.]: John Wiley and Sons, 2003.

SOMMERVILLE, I. **Engenharia de Software**. São Paulo, Brasil: Addison-Wesley, 2003.

STUDDARD, D.; PINTO, M. **The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws**. 2nd. ed. Indianapolis, USA: Wiley Publishing, 2011.

SULLO, C.; LODGE, D. **Nikto Web Server Security Scanner**. 2017. Disponível em: <https://cirt.net/Nikto2>. Acesso em: 16 abr. 2017.

TANENBAUM, A. S. **Redes de computadores**. 5th. ed. São Paulo, Brasil: Pearson Prentice Hall, 2011.

Tenable Network Security. **Nessus Vulnerability Scanner**. 2017. Disponível em: <https://www.tenable.com/products/nessus-vulnerability-scanner>. Acesso em: 16 abr. 2017.

THE INTERNET ENGINEERING TASK FORCE. **RFC 2616: Hypertext Transfer Protocol – HTTP/1.1**. 1999. Disponível em: <https://tools.ietf.org/html/rfc2616>. Acesso em: 15 abr. 2017.

THE INTERNET ENGINEERING TASK FORCE. **RFC 2828: Internet Security Glossary**. 2000. Disponível em: <https://www.ietf.org/rfc/rfc2828.txt>. Acesso em: 15 abr. 2017.

The OWASP Foundation. **OWASP Top 10 - O dez riscos de segurança mais críticos em aplicações web**. 2013. Disponível em: <https://www.owasp.org/images/9/9c/OWASP_Top_10_2013_PT-BR.pdf>. Acesso em: 4 mai. 2017.

UTO, N. **Teste de Invasão em Aplicações WEB**. Rio de Janeiro, Brasil: Escola Superior de Redes, 2013.

Verizon e Serviço Secreto dos Estados Unidos. **2010 DATA BREACH INVESTIGATIONS REPORT**. 2010. Disponível em: <http://www.verizonenterprise.com/resources/reports/rp_2010-data-breach-report_en_xg.pdf>. Acesso em: 17 abr. 2017.

VIVO, M. de et al. A review of port scanning techniques. **Acm Sigcomm Computer Communication Review**, New York, USA, v. 29, n. 2, p. 41–48, 1999.

VOGT, P. et al. Cross-site scripting prevention with dynamic data tainting and static analysis. In: NDSS, 2007, San Diego, US. **Proceedings of Network and Distributed System Security**. [S.l.]: IEEE, 2007.

Whitehat Security. **WEB APPLICATIONS SECURITY STATISTICS REPORT 2016**. 2010. Disponível em: <https://info.whitehatsec.com/rs/675-YBI-674/images/WH-2016-Stats-Report-FINAL.pdf>. Acesso em: 17 abr. 2017.

WIESMANN, A. et al. **A Guide to Building Secure Web Applications and Web Services**. 2nd. ed. [S.l.]: The OWASP Foundation, 2005.

WPScan Team. **WPScan Black Box WordPress Vulnerability Scanner**. 2017. Disponível em: <https://wpscan.org/>. Acesso em: 16 abr. 2017.

XSSER. **XSSer: Cross Site "Scripter"**. 2017. Disponível em: <https://xsser.03c8.net/>. Acesso em: 4 mai. 2017.

YU, D. et al. Javascript instrumentation for browser security. In: **Proceedings of the 34th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages**. New York, USA: ACM, 2007. p. 237–249.

ANEXO A — TRABALHO DE GRADUAÇÃO - I

- Título: O Processo de Testes de Intrusão do CPD-UFRGS
- Data: Novembro de 2016

O Processo de Testes de Intrusão do CPD-UFRGS

Pablo Caiã de Mello Soares[1], **Raul Fernando Weber**[1]

[1]Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brazil

{pcmsoares, weber}@inf.ufrgs.br

Abstract. *Vulnerabilities found in web applications are widely used by attackers to steal information and to break into computer networks. The fact that they are exposed to millions of users over the Internet shows that security is an essential requirement in such applications project. From intrusion tests, it is possible to discover vulnerabilities and then correct them, constituting a preventive action to invasions. The purpose of this work is to expose how these tests are applied to the sites of UFRGS by CPD, formalizing a process for this activity and analyzing its effectiveness through case studies.*

Resumo. *Vulnerabilidades presentes em aplicações web são amplamente utilizadas por atacantes para o roubo de informações e invasões a redes de computadores. O fato de estarem expostas a milhões de usuários pela Internet evidencia que segurança é um requisito indispensável no projeto de tais aplicações. A partir de testes de intrusão, é possível descobrir vulnerabilidades e posteriormente corrigi-las, constituindo-se de uma ação preventiva a invasões. A proposta deste trabalho é expor como estes testes são aplicados aos sites da UFRGS pelo CPD, formalizando um processo para esta atividade e analisando sua efetividade através de estudos de caso.*

1. Introdução

Teste de intrusão, também chamado de teste de invasão, teste de penetração ou pentest, é um método utilizado para verificar a segurança de um ambiente, plataforma ou sistema, por meio da simulação de ataques reais explorando as vulnerabilidades encontradas (UTO 2013). Quando o ambiente para testes se restringe a aplicações web, este método deve ser efetuado de forma mais específica, considerando as características inerentes a esses sistemas (MEUCCI et al. 2015).

O fato de tais aplicações estarem conectadas à Internet as tornam inseguras por padrão, pois está sendo permitido o acesso direto de desconhecidos ao servidor hospedeiro (BALLAD et al. 2009). Caso exista um firewall, por exemplo, excessões precisam ser criadas para acessos externos serem permitidos neste servidor, que usualmente fazem acesso a outros servidores da rede interna que costumam não ser acessíveis diretamente pela Internet, ou seja, a segurança de todo ambiente é afetada para tornar a aplicação disponível. Portanto, se uma aplicação for insegura, atacantes podem ganhar acesso e controle de servidores de uma rede através de suas vulnerabilidades, evitando medidas de segurança física e de rede (BALLAD et al. 2009).

Para validar a segurança de uma aplicação web e se prevenir de ataques, cuja as motivações mais comuns são financeiras, de prestígio, de demonstração de poder, ideológicas ou comerciais (CERT-BR, 2012), o guia de testes da OWASP[1] (MEUCCI et al. 2015) estabelece um arcabouço que abrange e defende a necessidade de diversos tipos de testes de segurança diferentes conforme a fase do ciclo de vida do desenvolvimento de um software. O mesmo guia sugere que testes de intrusão sejam realizados na fase de implantação[2].

No âmbito da UFRGS, para testar a segurança de sites desenvolvidos pelas diversas unidades da Universidade, o Departamento de Segurança da Informação (DSInf) do Centro de Processamento de Dados (CPD), unidade responsável por prover serviços de TI (Tecnologia da Informação), utiliza de testes de intrusão para realizar tal tarefa. A metodologia de testes de segurança utilizada se resume aos testes de intrusão pelo fato de não existir, por diversos fatores, um acompanhamento dos projetos de aplicações web desde sua concepção e a maioria dos sites estarem hospedados nas próprias unidades, o que dificulta a obtenção de códigos-fonte, impossibilitando o uso de técnicas como revisão de código, por exemplo.

A necessidade de haver a auditoria de segurança se justifica pelo fato de que não existir um acompanhamento por parte do CPD em relaçao a aplicação deste requisito no projeto e desenvolvimento desses sistemas, feito por bolsistas, funcionários ou empresas terceirizadas, em que não se há garantia sobre suas noções de desenvolvimento seguro. Infelizmente, tal problema entra de acordo com Wiesmann et al. (2005), que afirma não existir ainda na maioria das empresas e projetistas de software, uma movimentação forte em prol da segurança de TI.

A responsabilidade que o DSInf possui para auditar estes sites e averiguar se estão em conformidade com a Política de Segurança da Informação (PSI) da UFRGS (CONSUN, 2014) é garantida pelo item III do Art. 5º da PSI, que define a seguinte atividade: *"monitorar, auditar e avaliar periodicamente as práticas de segurança da informação adotadas pela Universidade"*. Neste documento, ainda é garantido, através de um parágrafo único, o direito de solicitar por medidas para garantir que estas práticas sejam seguidas: *"Cabe às demais unidades da Universidade, no âmbito de suas competências, a implementação e o acompanhamento de ações para segurança da informação"*.

Essa primeira parte do trabalho consiste na apresentação das principais técnicas existentes para o teste de segurança em aplicações e, de forma mais aprofundada, sobre testes de intrusão visando a avaliação de aplicações web. Na segunda parte do trabalho, será apresentado um processo de testes adaptado ao ambiente do CPD da UFRGS baseado em testes de intrusão, justificando a escolha desta técnica e todas adaptações e decisões feitas em cada etapa do processo, também será justificado os tipos de vulnerabilidades priorizadas e ferramentas utilizadas nos testes. Por fim, esse processo será testado em estudos de caso e será feita uma analise da sua eficácia e de possíveis melhorias.

Este trabalho é organizado como se segue. A seção 2 introduz as principais técnicas de testes de segurança para aplicações. A seção 3 descreve a metodologia de

[1]The Open Web Application Security Project: https://www.owasp.org/

[2]fase do ciclo de vida de um software, no contexto de um Sistema de Informação, que corresponde textualmente à passagem do software para a produção.

um teste de intrusão, introduzindo alguns conceitos envolvidos, as etapas que envolvem este tipo de teste e ferramentas que podem ser utilizadas para auxiliar o processo. A seção 4 explica de forma sucinta a metodologia e os objetivos da segunda parte do trabalho. A seção 5 faz as considerações finais. Na seção 6 consta a bibliografia consultada.

2. Técnicas de teste de segurança

O objetivo desta seção, que tem como base o guia de testes da OWASP - *"OWASP Testing Guide 4.0"* (MEUCCI et al. 2015), é apresentar as principais técnicas existentes de teste de segurança em aplicações, para posteriormente poder entender, de forma mais detalhada, a escolha do teste de intrusão como forma de verificar a segurança dos sites da Universidade. Segundo os autores deste guia, cada uma destas técnicas possuem suas características como conhecimento técnico necessário para efetuar o teste, tempo de execução, a partir de que fase da SDLC[3] pode ser aplicado e suas vantagens e desvantagens.

2.1. Inspeção e revisão manual

Inspeções manuais são avaliações que normalmente testam as implicações em segurança de pessoas, políticas e processos, que também podem incluir a inspecção de decisões de tecnologia tais como design arquitetural. Estes procedimentos são geralmente conduzidos por análise de documentação ou na realização de entrevistas com os desenvolvedores ou os proprietários do sistema.

Embora o fato dos conceitos de inspeção e revisão manual serem simples, estas técnicas podem estar entre as mais potentes e eficazes disponíveis (MEUCCI et al. 2015). Quando se pergunta a alguém como certa funcionalidade opera e porque foi implementada de uma maneira específica, é possível determinar se quaisquer preocupações de segurança são suscetíveis de serem evidentes.

Revisões manuais são particularmente boas para testar se as pessoas envolvidas em um projeto de software compreendem o processo de segurança, se foram informados da política, e se possuem as habilidades necessárias para projetar ou implementar uma aplicação segura (MEUCCI et al. 2015). Outras atividades, incluindo a revisão da documentação, das políticas de desenvolvimento seguro, requisitos de segurança e design da arquitetura da aplicação, devem todos ser realizados utilizando inspeções manuais.

2.2. Modelagem de ameaças

Modelagem de ameaças é uma técnica utilizada para se identificar ameaças de segurança que sistemas e aplicações podem vir a enfrentar. Esta técnica geralmente é utilizada por designers de sistemas com conhecimentos em segurança e pode ser vista como uma avaliação de riscos para aplicações (MEUCCI et al. 2015). Modelagem de ameaças permite que se possa desenvolver estratégias de mitigação de potenciais vulnerabilidades e pode ser aplicada desde o início do projeto de uma aplicação.

Para desenvolver um modelo de ameaça, existe a abordagem proposta pelo o padrão NIST 800-30 (NIST, 2012) para avaliação de risco. Esta abordagem envolve ações como:

[3]Software Development Life Cycle

- Decompor a aplicação para entender seu funcionamento, seus ativos e conectividade;
- Definir e classificar ativos, de acordo com a importância do ativo para o negócio e tangibilidade;
- Explorar potenciais vulnerabilidades, sejam elas técnicas, operacionais ou de gerenciamento;
- Explorar potenciais ameaças planejando possíveis vetores de ataques vistos da perspectiva do atacante;
- Criar estratégias de mitigação, desenvolver controles de mitigação para cada ameaça que se considerar real.

2.3. Revisão de código

Revisão de código fonte é o processo em que se verifica o código de uma aplicação por questões de segurança. Esta técnica é considerada a técnica mais efetiva para se detectar vulnerabilidades nas fases iniciais do desenvolvimento de uma aplicação (CONKLIN et al. 2016), além de muitas vulnerabilidades graves não poderem ser detectadas com qualquer outra forma de teste (MEUCCI et al. 2015). O guia de testes da OWASP cita o fato da maioria dos especialistas em segurança concordarem que nao há um método equivalente que substitua a revisão de código pois todas as informações para a identificação de problemas de segurança estão no código em algum lugar.

Diversos problemas de segurança são mais difíceis de se descobrir com outras técnicas de segurança, tais como teste de intrusão (CONKLIN et al. 2016). No entanto, com o código-fonte, um analista pode determinar com precisão o que está acontecendo e remover o trabalho de adivinhação de um teste de intrusão, no entanto, usualmente tal técnica costuma ser mais demorada (BIRD, 2013).

Exemplos de problemas que são particularmente propícios de serem encontrados através da revisão de código fonte incluem problemas de concorrência, falha na lógica de negócios, problemas de controle de acesso e criptografia fraca, bem como backdoors, trojans e outras formas de código malicioso (CONKLIN et al. 2016). Análise de código fonte também pode ser extremamente eficiente para encontrar problemas de implementação, como áreas do código onde a validação de entrada não é realizada ou quando os procedimentos de controle são falhos. (MEUCCI et al. 2015)

2.4. Teste de intrusão

Teste de intrusão tem sido a prática de segurança mais comum e frequente de ser aplicada (MCGRAW 2006). Segundo Meucci et al. (2015), o trabalho envolvido nesta técnica consiste na ação de testar uma aplicação em execução remotamente para encontrar vulnerabilidades de segurança, sem conhecer seu funcionamento interno. Normalmente, a equipe de teste têm acesso a uma aplicação como se fossem usuários, opcionalmente pode ser fornecido algum nível de acesso ao sistema.

Apesar desta técnica ser eficaz para testar a segurança de redes, isto não ocorre naturalmente quando se trata de aplicações (MEUCCI et al. 2015). Quando a técnica é realizada em redes e infrastruturas, a maioria do trabalho está envolvido em procurar e posteriormente explorar vulnerabilidades conhecidas em tecnologias específicas, pois normalmente são compostas por produtos *"off-the-shelf"*[4] com configurações mais ou

[4]Produtos de prateleira, soluções prontas adquiridas comercialmente.

menos padrões (STUDDARD et al. 2011). Como as aplicações web tendem a ser únicas, este trabalho deve ser mais especializado. Existem ferramentas que foram desenvolvidas para automatizar o processo, mas com a natureza das aplicações web sua efetividade geralmente é baixa (UTO 2013).

3. Metodologia de um teste de intrusão

O objetivo desta seção é apresentar os conceitos envolvidos e a organização de um teste de intrusão, os quais serão relevantes para o desenvolvimento deste trabalho. Aborda-se sobre os conceitos de intrusão, tipos de teste de intrusão, além da especificação das etapas do processso conforme o recomendado pela literatura da área.

3.1. Conceitos sobre intrusão

Para um melhor entendimento dos diversos termos relacionados a testes de intrusão utilizados no decorrer deste trabalho, será introduzido nesta seção os conceitos básicos sobre vulnerabilidade, ameaça e ataque. As definições são dadas pela RFC 2828 - Internet Security Glossary (IETF, 2000).

Vulnerabilidade é definida como uma falha, ou fraqueza, no design ou no gerenciamento de um sistema computacional, que quando explorada por um atacante resulta na violação da segurança deste sistema. Tipicamente, uma vulnerabilidade é um ponto de entrada para o sistema.

Uma ameaça consiste na possibilidade de violação, de forma acidental ou proposital, de uma vulnerabilidade em um sistema computacional causando danos. Os danos são causados por ataques.

Ataque é o acesso, não autorizado, do sistema computacional. Tipicamente, um ataque ocorre quando um atacante, utilizando-se de uma vulnerabilidade, tenta executar ações maliciosas, como invadir um sistema, acessar informações confidenciais, disparar ataques contra outros computadores ou tornar um serviço inacessível (CERT-BR, 2012). Os ataques podem ser ativos ou passivos e podem ser iniciados de dentro do perímetro de segurança (insider) ou do exterior do perímetro de segurança (outsider). Em um ataque passivo, tenta-se obter ou fazer uso de informações do sistema. Enquanto em um ataque ativo, existe a tentativa de alterar recursos do sistema ou afetar seu comportamento.

3.2. Tipos de teste

Testes de intrusão, de acordo com Uto (2013), podem ser classificados de acordo com a quantidade de informação que é disponibilizada ao testador. Desta definição, as três seguintes terminologias são mais comuns de serem utilizadas para esta classificação, segundo Muniz et al. (2013):

- **Teste caixa-preta:** visa simular as mesma condições de um atacante real, que possui acesso somente às informações públicas do alvo, como endereços IPs externos, nomes de domínio e ramo de negócio da entidade.
- **Teste caixa-branca:** neste tipo de teste, todas as informações do alvo são fornecidas ao testador, incluindo topologia de rede, plataformas utilizadas, diagramas de casos de uso, linguagens empregadas, códigos-fonte e endereçamento interno. O objetivo desta aborgagem é simular ataques que podem ser realizados por pessoas com conhecimento privilegiado do alvo, como funcionários e ex-colaboradores.

- **Teste caixa-cinza:** este tipo de teste é uma mistura dos dois anteriores, antecipando ao auditor aquelas informações do alvo que um atacante obteria, cedo ou tarde, em um processo de invasão real. Desse modo, o objetivo desta modalidade é acelerar a execução do teste, poupando o tempo gasto pelo auditor nos processos de reconhecimento e mapeamento.

Existe também a possibilidade de considerar mais uma variável na classificação de testes, A metodologia OSSTMM (HERZOG, 2010) considera se a equipe de segurança do ambiente alvo está consciente da realização dos testes de segurança pelo testador. Com a adição desta variável, é possível enumerar mais três tipos de testes:

- **Teste duplo-cego:** é um teste caixa-preta, no qual a equipe de segurança do não é avisada sobre a realização dos testes.
- **Teste duplo-cinza:** é um teste de caixa-cinza em que a equipe de segurança do ambiente alvo sabe o período e escopo dos testes, mas não os vetores ou canais que serão empregados pelo auditor.
- **Teste reverso:** é um teste em que o testador recebe todas as informações disponíveis sobre o alvo e no qual a equipe de segurança do ambiente alvo não é notificada sobre a realização dos testes.

3.3. Etapas de um teste de intrusão

Testes de intrusão são uma ferramenta importante no desenvolvimento e manuntenção de uma aplicação segura (MCGRAW, 2006), de forma a evitar que vulnerabilidades sejam inseridas no ambiente produto e avaliar a susceptibilidade dos sitemas a novos tipos de ataque. É fundamental realizar este processo de acordo com métodos pré-definidos, de modo a permitir que diferentes pessoas alcancem resultados parecidos, que possam ser reproduzidos (UTO, 2013).

Um teste de intrusão é um processo cíclico que envolve a execução de diversas atividades, conforme o ilustrado na figura 1. Neste artigo, as divisões das etapas deste processo, assim como suas definições, são as dadas por Uto (2013), que as adapta ao contexto de aplicações.

3.3.1. Definição de escopo e autorização

Para se iniciar um teste de intrusão, é necessário se obter do solicitante uma autorização para execução do teste, o escopo que será coberto pela atividade, objetivos e duração dos testes (BACUDIO et al. 2011). Usualmente, isso é feito através de um contrato assinado entre as partes, que além de selar um compromisso, define o tipo do teste, as premissas do trabalho e as informações e privilégios que serão fornecidos pelo contratantes, no caso do teste ser caixa-branca ou cinza (UTO, 2013).

3.3.2. Reconhecimento

Nesta etapa, todas as ações são direcionadas a levantar o máximo de informação sobre uma aplicação web que for possível e tentar entender sua lógica de funcionamento (BACUDIO et al. 2011). Usualmente, o auditor busca identificar servidores que suportam

a aplicação, sistemas operacionais instalados, linguagens de programação empregadas, nomes e potenciais usuários do sistema, configurações de softwares, entre outros (UTO, 2013). Todas essas informações são obtidas de diversas maneiras, como testes no próprio ambiente ou pesquisas em fontes de informações externas.

3.3.3. Mapeamento

No mapeamento, todas informações que foram coletadas nas etapas anteriores devem ser relacionados para se criar um mapa da aplicação, que reflita toda sua estrutura, contendo os arquivos componentes, funcionalidades, tecnologias utilizadas e pontos de entrada de informação (UTO, 2013). Esse processo é realizado se obtendo uma cópia das partes acessíveis da aplicação, para então identificar os pontos de entrada de informação e relacionar o que se foi obtido com as informações de reconhecimento da etapa anterior.

3.3.4. Identificação de vulnerabilidades

Após se obter um mapa construído da aplicação, o próximo passo consiste em descobrir, caso existam, vulnerabilidades presentes nas camadas que compõem a aplicação. O guia de testes do OWASP (MEUCCI et al. 2015) expõe uma compilação de verificações para dezenas de vulnerabilidades, agrupadas em classes. Alguns exemplos destes agrupamentos se apresentam como vulnerabilidades relacionadas à levantamento de informações, gerenciamento de configuração, validação de dados, entre tantos outros. De acordo com Palmer (2007), vulnerabilidades encontradas em aplicações web podem ser agrupadas em três famílias:

- Vulnerabilidades presentes em servidores web, que envolve a varredura por vulnerabilidades conhecidas ou, de forma não tão frequente, a descoberta de novas vulnerabilidades nesses servidores;
- Vulnerabilidades devido a scripts CGI (Common Gateway Interface) e páginas padrões expostas, que basicamente são identificadas a partir de varreduras por páginas padrões e CGIs anteriormente reconhecidos como inseguros presentes em um servidor;
- Vulnerabilidades existentes na aplicação web em sí, identificadas a partir do teste de pontos de entrada de informação existentes e das funcionalidades providas pela aplicação.

A identificação de vulnerabilidades se dá através de testes manuais ou automáticos. Para testes automáticos, podem ser utilizados ferramentas como scanners de vulnerabilidades, que têm por objetivo encontrar, automaticamente, o maior número de vulnerabilidades existentes em um ativo (UTO, 2013). Infelizmente, scanner de vulnerabilidades em aplicações possuem limitações (DOUPE, 2010). Segundo Studdard et al. (2011), tais ferramentas trabalham a partir da análise sintática das respostas da aplicação e identificação de mensagens de erros comuns, códigos de estado HTTP e conteúdo suprido por usuários sendo copiado para páginas, não sendo capazes de entender o significado semântico do conteúdo analisado. Scanners de vulnerabilidade não conseguem detectar, por exemplo, que a modificação de um parâmetro que determina o preço de um produto pelo usuário é uma vulnerabilidade.

3.3.5. Exploração de vulnerabilidades

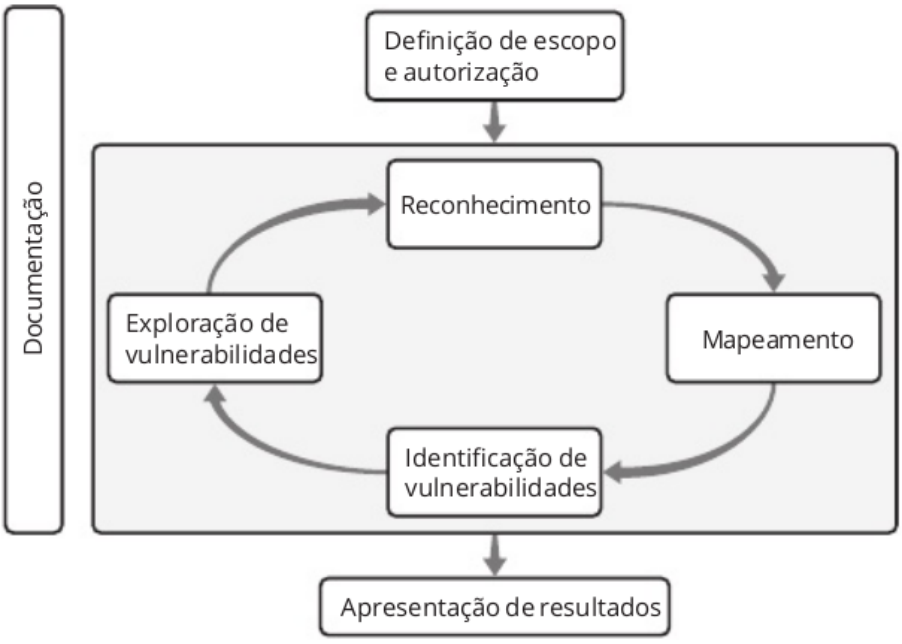
A exploração das vulnerabilidades encontradas, quando possível, é a última fase do ciclo e visa tentar violar a segurança da aplicação (UTO, 2013). O foco desta etapa é estabelecer acesso a um sistema ou recurso evitando restrições de segurança (PTES, 2014). Normalmente, após a finalização desta etapa, é esperado que se tenha obtido um maior nível de acesso ao sistema ou uma melhor compreensão dele, de forma a permitir uma abordagem por outro ângulo (UTO, 2013).

3.3.6. Documentação e apresentação de resultados

O processo de documentação ocorre paralelamente ao ciclo de testes, sendo empregado desde o contrato formal (UTO, 2013). Normalmente deve estar incluido na documentação, de forma detalhada, o máximo de informações sobre o ataque, como pré-condições, ferramentas utilizadas e métodos empregados, sendo possível a reprodução da exploração da vulnerabilidades caso seja desejado por quem solicitou o teste. Usualmente é incluído também recomendações apontando soluções para os problemas encontrados.

Conforme Uto (2013), os resultados encontrados devem ser apresentados aos clientes, de forma a ser entendível de acordo com a audiência. Detalhes técnicos das explorações devem ser reservados ao corpo técnico da empresa ou instituição. No caso da audiência ter um perfil mais administrativo, os resultados apresentados devem focar nos impactos resultantes dos possíveis ataques.

Figura 1: Etapas de um teste de intrusão



Fonte: (UTO, 2013).

3.4. Ferramentas

Na realização de um teste de intrusão, é interessante o uso de ferramentas, por existir a possibilidade de elevar a produtividade do processo. Tais ferramentas são classificadas de acordo com suas funcionalidades de forma comum à diversos autores da área como Uto (2013) e Palmer (2007), Muniz et al. (2013), existindo a possibilidade de se encaixarem em mais de uma classificação e etapa do processo de testes.

3.4.1. Navegadores web

Navegadores são softwares utilizados por usuários para poder explorar e interagir com sistemas na Internet. Cada navegador disponível possui suas peculiaridades com o protocolo HTTP e da forma como documentos HTML são interpretados. Do ponto de vista do atacante, tais diferenças implicam que alguns tipos de ataques não funcionam em todos navegadores, constuindo-se uma boa pratica realizar testes considerando diversos navegadores em mais de uma versão (UTO, 2013). Exemplo mais populares desta categoria podem ser o Internet Explorer, Google Chrome, Mozilla Firefox, Apple Safari e Opera.

3.4.2. Proxies de interceptação

Proxies de interceptação são ferramentas utilizadas para inspecionar requisições e respostas HTTP/HTTPS, permitindo a possibilidade de manipulação de suas informações (STUDDARD et al. 2011). Tais ferramentas usualmente roda localmente, na mesma estação em que se encontra o navegador web, interceptando e exibindo todo conteúdo dos pacotes HTTP/HTTPS gerados pela navegação (UTO, 2013).

Além da funcionalidade básica de interpretação, estas ferramentas podem apresentar diversas funcionalidades como filtros para seleção de mensagens, manuntenção de histórico de requisições e respostas interceptados, manipulação das interceptações diretamente do navegador, entre outros (STUDDARD et al. 2011). Exemplos desta ferramenta seriam o WebScarab, da OWASP, ProxyStrike, Paros e Burp Suite, que é uma suíte de testes paga, mas que disponibiliza seu proxy na versão gratuíta.

3.4.3. Web spiders

Web Spiders são ferramentas que automatizam o processo de se obter todo conteúdo navegável de um site (PALMER, 2007). Essas ferramentas funcionam a partir da requisição de páginas, fazendo o parsing do seu conteúdio na procura de links para acessar outras páginas e continuar o processo recursivamente (STUDDARD et al. 2011).

Além da função básica, web spiders podem ainda analisar formulários HTML e submete-los à aplicação com valores randômicos para extender a capacidade de navegação entre páginas e fazer o parsing de JavaScript para extração de URLs (UTO, 2013). Alguns exemplos de web spiders ou de ferramentas comn tal funcionalidade incluem Burp Suite, OWASP ZAP e Web Scarab.

3.4.4. Scanners de portas e serviços

Scanner de portas são utilizados para determinar que portas em um host podem estar sendo escutadas por processos esperando por possíveis conexões (DE VIVO et al. 1999). Adicionalmente, tais ferramentas são capazes de identificar os serviços específicos que estão escutando essas portas, além de opcionalmente poderem detectar o tipo e versão do sistema operacional utilizado (UTO, 2013). Segundo De Vivo et al. (1999), tais portas podem caracterizar a quantidade de exposição de hosts para potenciais ataques.

Estas ferramentas são utilizadas por analistas de segurança para verificar se os ativos zelados estão executando apenas os serviços autorizados, e para levantar informações preliminares do ambiente de teste de intrusão e varredura de vulnerabilidades (UTO, 2013). O software NMAP é um exemplo desta categoria, sendo capaz de detectar, aplicações, serviços e sistemas operacionais (MUNIZ et al. 2013).

3.4.5. Scanners de vulnerabilidades

Ferramentas desta categoria, de forma mais específica, podem se concentrar ou incluir a funcionalidade de detectar vulnerabilidades no servidor web (PALMER, 2007), detectando arquivos e diretórios instalados por padrão ou conhecidos como vulneráveis, problemas específicos de versão e de configuração como Webshag, nikto e Skipfish. Existem scanners para reconhecer vulnerabilidades em plataformas web específicas como CMSs, alguns exemplos são o JoomScan para Joomla, WPScan para WordPress e o DPScan para Drupal.

Para a aplicação web em sí, existem ferramentas que possuem a funcionalidade de mapear a aplicação para analisar o conteúdo de páginas e testar pontos de entrada de informação (DOUPÉ, 2010). Este tipo de varredura depende, em partes, da qualidade do mapeamento para ser eficiente e é limitada a detectar um conjunto limitado de vulnerabilidades possíveis em aplicações web (UTO, 2013). Algumas ferramentas disponíveis são OWASP ZAP, Burp Suite, w3af, vega, Wapiti e Arachni.

Existem scanners com a função de detectar e identificar vulnerabilidades em sistemas operacionais, aplicações instaladas e demais programas (MUNIZ et al. 2013), complementando a funcionalidade de varredores de portas e serviços. Estas ferramentas se baseiam na busca de vulnerabilidades conhecidas (STUDDARD et al. 2011) e alguns exemplos desta categoria incluem Nessus, OpenVAS e a engine de scripting do NMAP.

3.4.6. Exploradores de vulnerabilidades

Ferramentas que possuem a funcionalidade de explorar vulnerabilidades são úteis para determinar o esforço necessário para explorar potencias vulnerabilidades e julgar o risco associado caso haja um ataque (MUNIZ et al. 2013). Alguns exemplos podem ser o sqlmap, fimap e Beef para explorar vulnerabilidades específicas ou ferramentas mais completas como w3af e Metasploit, que possuem a capacidade de explorar diversos tipos de vulnerabilidades, além de outras funcionalidades.

4. Metodologia e organização

Na sequência deste trabalho, será feito um levantamento de requisitos que ferramentas disponíveis precisam possuir para poderem ser utilizadas no processo de testes. Serão selecionadas as ferramentas que mais se adequarem a estes requisitos para então ser feito um estudo mais profundo de como suas funcionalidades podem auxiliar a metodologia de testes.

A partir do momento que já existir algum resultado do estudo sobre ferramentas até sua conclusão, estes serão adequados ao processo de testes de intrusão utilizado pelo DSInf no CPD. Esta adaptação será feita a partir da análise de diversos pontos como a influência da infraestrutura de rede da Universidade nos testes, das informações iniciais disponíveis dos alvos e das tecnologias e funcionalidades mais comuns de serem encontradas nos sites da Universidade. Será justificado a escolha do teste de intrusão para avaliar a segurança dos sites em detrimento de outras tecnicas, além da adequação das etapas da técnica escolhida apresentadas neste artigo. Também tentará ser feito uma divisão dos testes executados na etapa de identificação de vulnerabilidades em categorias de prioridade de acordo com critérios como: dificuldade de execução, frequência da vulnerabilidade testada e complexidade de correção.

No momento em que for obtido uma metodologia aplicável, mesmo que incompleta, se iniciarão os testes nos sites para se obter os estudos de caso que avaliarão a eficiência do processo e que será possível apontar melhorias a partir de falhas observadas. O motivo de aplicar a metodologia, mesmo que em fase de construção, será justamente para avaliar as ferramentas escolhidas e adaptações feitas, sendo possível otimizar todo trabalho feito até se obter um conjunto de ferramentas eficientes e uma metodologia completa. Os resultados da aplicação do processo serão avaliados para analisar sua eficiência.

Para a realização das atividades planejadas, é proposto o seguinte cronograma:

1. Estudo de ferramentas mais adequadas para auxiliar os testes.
2. Adaptação do processo de teste de intrusão ao ambiente do CPD.
3. Aplicação do processo para se obter estudos de caso.
4. Avaliação dos resultados.
5. Redação da segunda parte deste trabalho.
6. Apresentação do Trabalho de Graduação 2.

Tabela 1: Cronograma para a segunda parte do Trabalho de Graduação

Tarefa	2017					
	Janeiro	Fevereiro	Março	Abril	Maio	Junho
1	X	X				
2		X	X			
3		X	X	X		
4				X	X	
5	X	X	X	X	X	X
6						X

Fonte: Próprio autor.

5. Considerações finais

Este artigo teve o objetivo de introduzir a técnica de teste de intrusão como forma de testar a segurança dos sites da Universidade. Outras técnicas foram brevemente apresentadas para demonstrar que existem outras formas de se realizar esta tarefa, que posteriormente, no sequência deste trabalho, serão justificados os motivos de não serem aplicadas. Tal procedimento é relevante pelo fato de aplicações web, de uma maneira geral, proverem soluções e funcionalidades de uma forma facilitada e centralizada para os usuários, no entanto, tais incrementos tendem a trazer problemas se não forem planejados e implementados de forma adequada.

Ainda que existam, na literatura, modelos de processo para testes de intrusão, com etapas bem definidas e sugestão de ferramentas para serem utizadas e procedimentos a serem seguidos, o ambiente em que estes testes se aplicam também deve ser considerado. Diferentemente de uma empresa que oferece serviços de testes de segurança, com ambientes distintos conforme o contrato, o DSInf se restringe a testar os sites e servidores sob sua responsabilidade, permitindo adequar este processo as peculiaridades, padrões e restrições impostos. A intenção da segunda parte do trabalho é justamente realizar a adaptação do teste de intrusão para o ambiente da Universidade e avaliar sua efetivadade, sem expor nocivamente sua rede e demais ativos.

Para isso, é necessário que, além de consultas a livros e artigos sobre o assunto, seja explorado uma experiência consideravelmente empírica, com a aplicação da técnica constantemente no decorrer deste trabalho, possibilitando enxergar formas de realizar a adaptação proposta da melhor forma possível. Isso significa que a documentação para a continuação deste trabalho necessitará constantes revisões devido ao seu perfil prático do qual a experiência adquirida pode influenciar durante todo seu desenvolvimento, que resultará em um modelo que muito provavelmente poderá ser aproveitado por outras pessoas que se encarregarem de tal tarefa.

6. Bibliografia

UTO, N. **Teste de Invasão de Aplicações Web**, Rio de Janeiro, Brasil, RNP/ESR, 2013.

BALLAD, T.; BALLAD, W. **Securing PHP Web Applications**, [s.l.], Addison-Wesley, 2009.

MEUCCI, M.; MULLER, A. **OWASP Testing Guide 4.0**, The OWASP Foundation, 2015.

CONKLIN, L.; ROBINSON, G. **Code Review Guide 2.0 Alpha Release**, The OWASP Foundation, 2016.

HERZOG, P. **OSSTMM 3 - The Open Source Security Testing Methodology Manual**, ISECOM, 2010.

CONSELHO UNIVERSITÁRIO DA UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL. **Política de Segurança da Informação da UFRGS.** 2014. Disponível em: http://www.ufrgs.br/consun/legislacao/documentos/. Acesso em: 15 set. 2016.

LAWTON, G. **Web 2.0 Creates Security Challenges**. San Francisco, EUA, 2007.

BIRD, T. **Choosing Between a Penetration Test and a Secure Code Review**, disponível em: https://dzone.com/articles/choosing-between-penetration. Acesso em: 15 set. 2016.

CENTRO DE ESTUDOS, TRATAMENTO E RESPOSTA DE INCIDENTES DE SEGURANÇA NO BRASIL. **Cartilha de Segurança para Internet.** 2012. Disponível em: http://cartilha.cert.br/. Acesso em: 20 out. 2016.

MUNIZ, J.; LAKHAMI, A. **Web Penetration Testing with Kali Linux**. Birmingham, UK, Packt Publishing, 2013.

PALMER, S. et al. **Web Application Vulnerabilities:** Detect, Exploit, Prevent. Burlington, USA, Syngress, 2007.

STUDDARD, D.; PINTO, M. **The Web Application Hacker's Handbook:** Finding and Exploiting Security Flaws. 2 ed. Indianopolis, USA, Wiley Publishing, 2011.

THE INTERNET ENGINEERING TASK FORCE. **RFC 2828:** Internet Security Glossary. 2000. Disponível em: https://www.ietf.org/rfc/rfc2828.txt. Acesso em: 26 out. 2016.

PENETRATION TESTING EXECUTION STANDARD **Penetration Testing Execution Standard.** Disponível em: http://www.pentest-standard.org/index.php/Intelligence_Gathering. Acesso em: 10 set. 2014.

DOUPÉ, A.; COVA, M.; VIGNA, G. **Why Johnny Can't Pentest:** An analysis of Black-Box Web Vulnerability Scanners.

MCGRAW, G. **Software Security:** Building Security In. 1 ed. Addison-Wesley. 2006.

NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY. **Risk management guide for information technology systems.** 2012. Disponível em: http://csrc.nist.gov/publications/nistpubs/800-30-rev1/sp800_30_r1.pd. Acesso em: 27 out. 2016.

BACUDIO, Aileen et al. An Overview About Penetration Testing. **International Journal Of Network Security & Its Applications.** p. 19-38. nov. 2011.

DE VIVO, M. et al. A Review of Port Scanning Techniques. **Acm Sigcomm Computer Communication Review**, New York, v. 29, n. 2, p.41-48, maio 1999.