JÚLIO CÉSAR DA SILVA PINTO

ESTUDO E EXPLORAÇÃO DE VULNERABILIDADES EM SISTEMAS MULTI-TENANTS

LAVRAS – MG

2019

JÚLIO CÉSAR DA SILVA PINTO

ESTUDO E EXPLORAÇÃO DE VULNERABILIDADES EM SISTEMAS MULTI-TENANTS

Monografia apresentada à Universidade Federal de Lavras, como parte das exigências para a obtenção do título de Bacharel em Sistemas de Informação.

Prof. Dr. Antônio Maria Pereira de Resende

Orientador

Prof. Dr. Victor Hugo Santiago Costa Pinto

Coorientador

LAVRAS – MG

2019

JÚLIO CÉSAR DA SILVA PINTO

ESTUDO E EXPLORAÇÃO DE VULNERABILIDADES EM SISTEMAS MULTI-TENANTS

Monografia apresentada à Universidade Federal de Lavras, como parte das exigências para a obtenção do título de Bacharel em Sistemas de Informação.

APROVADA em 27 de Novembro de 2019.

Prof. Dr. Antônio Maria Pereira de Resende	UFLA
Prof. Dr. Victor Hugo Santiago Costa Pinto	UFPA
Prof. Dr. Neumar Costa Malheiros	UFLA
Prof. Dr. Ricardo Ramos de Oliveira	IFSULDEMINAS

Prof. Dr. Antônio Maria Pereira de Resende
Orientador

Prof. Dr. Victor Hugo Santiago Costa Pinto
Co-Orientador

LAVRAS – MG
2019

AGRADECIMENTOS

Agradeço a minha família pelo apoio durante toda minha graduação.

A minha namorada Francielen por sempre estar ao meu lado, apesar da distância, pela compreensão e por sempre me apoiar e incentivar.

Aos professores do Departamento da Ciência da Computação da Universidade Federal de Lavras pelos ensinamentos dos últimos anos, especialmente aos professores Victor Hugo Santiago Costa Pinto e Antônio Maria Pereira Resende pela orientação, paciência, ensinamentos e auxílios no desenvolvimento e implementação do trabalho.

Aos outros membros da banca avaliadora— Dr. Neumar Costa Malheiros e Dr. Ricardo Ramos de Oliveira— pela disposição de ler meu estudo e participar da defesa do meu trabalho.

A todos os colegas discentes da graduação pela oportunidade de convivência e experiências compartilhadas.

A todos que de alguma forma colaboraram para a realização deste trabalho.

RESUMO

A computação em nuvem é um conjunto de abordagens e princípios que possibilita a entrega de infraestrutura, plataformas, aplicações e serviços sob demanda através da Internet. Devido à evolução da computação em nuvem, sistemas desenvolvidos sobre o modelo de entrega *SaaS* (*Software as a Service*) estão se tornando cada vez mais comuns no mercado. No entanto, a manutenção de sistemas ligeiramente distintos que compartilham muitos recursos em comum, pode ser uma tarefa dispendiosa e acarretar em muito retrabalho. A arquitetura *multi-tenant* possibilita o compartilhamento de uma única aplicação e de recursos computacionais para múltiplos clientes (*tenants* ou inquilinos), com o objetivo de reduzir custos e acelerar o desenvolvimento. Apesar do compartilhamento de recursos e da aplicação, essa arquitetura permite que cada *tenant* customize a aplicação para atender melhor as suas regras de negócio. Tais características de aplicações *multi-tenants* podem fazer com que as mesmas se tornem mais suscetíveis a vulnerabilidades e, consequentemente a ataques que podem comprometer dados dos *tenants*. Portanto, deve-se tomar especial cuidado na segurança da informação em aplicações que utilizam esta arquitetura. Apesar da evolução da computação em nuvem e da arquitetura *multi-tenant*, ainda não há muitos estudos na área de segurança da informação para sistemas desenvolvidos nessa arquitetura e existe desconfiança por parte das empresas que pretendem migrar seus sistemas legados. O objetivo deste trabalho concentra-se em obter evidências de possíveis vulnerabilidades nesse modelo de arquitetura, as quais podem comprometer a segurança em termos de exposição de dados. Por isto, testes de invasão (PENTEST) foram empregados em três aplicações que divergem em relação ao nível de isolamento de dados, denominadas: *iCardapio* é um sistema *multi-tenant* para disponibilização de cardápios eletrônicos; *MtShop* é um típico *e-commerce* que permite a criação de várias lojas virtuais e o *ToyExample* é uma aplicação desenvolvida neste trabalho para simular a vulnerabilidade de *SQL Injection*. Os resultados deste trabalho demonstraram que o comprometimento de um *tenant* ou a exploração de algumas vulnerabilidades podem causar danos à todos os *tenants*, devido à arquitetura *multi-tenant* das aplicações.

Palavras-chave: Computação em nuvem, *Software as a Service*, *Multi-tenant*, Segurança da informação, PENTEST.

ABSTRACT

Cloud computing comprises a set of approaches and principles that enables the on-demand delivery of infrastructure, platforms, applications and services over the Internet. Due to its evolution, systems developed under the SaaS (Software as a Service) delivery model have become increasingly common on the market. However, the maintenance of slightly different systems that share many resources may be a costly task and incur plenty of rework. The multi-tenant architecture enables the sharing of a single application and computing resources with multiple clients (tenants), towards reducing costs and accelerating development. However, each tenant can customize the application to best suit their business rules, which may make them more susceptible to vulnerabilities, hence, attacks that may compromise tenant data. Special care should be taken regarding information security in applications that use this architecture. Not many studies on the architecture in the area of information security for systems have been conducted, and some companies have shown distrust to migrate their legacy systems. This study aims at the obtaining of evidence of potential vulnerabilities in this architectural model that may compromise security regarding data exposure. Penetration tests (PENTEST) were applied in the following three applications of different levels of data isolation: iCardapio, a multi-tenant system that provides electronic menus, MtShop, a typical e-commerce that enables the creation of multiple online stores, and ToyExample, an application developed in this study that simulates SQL Injection vulnerability. The results show a jeopardized tenant, or exploitation of some vulnerabilities can damage all tenants due to the multi-tenant architecture of the applications.

Keywords: Cloud computing, Software as a Service, Multi-tenant, Information security, PENTEST.

LISTA DE FIGURAS

Figura 2.1 – Bancos de dados isolados 13
Figura 2.2 – Bancos de dados compartilhados, *schemas* isolados 14
Figura 2.3 – Bancos de dados e *schemas* compartilhados 15
Figura 2.4 – Visão geral de um sistema *SaaS multi-tenant* 16
Figura 2.5 – Sequência de um ataque *SQL Injection* 22
Figura 2.6 – Sequência de um ataque *XSS* 24
Figura 5.1 – Estrutura do GQM. 32
Figura 5.2 – Diagrama GQM do trabalho. 35
Figura 5.3 – Página inicial da aplicação *iCardapio*. 36
Figura 5.4 – Página inicial da aplicação *iCardapio*. 37
Figura 5.5 – Modal para cadastramento de produto no cardápio. 37
Figura 5.6 – Produto adicionado ao cardápio. 38
Figura 5.7 – Listagem do código da página inicial da aplicação *iCardapio*. 39
Figura 5.8 – Trecho do arquivo da biblioteca *jQuery*. 40
Figura 5.9 – Página *template* de cardápio. 41
Figura 5.10 – Vulnerabilidade encontrada na biblioteca *jQuery*. 42
Figura 5.11 – Vulnerabilidade encontrada pela ferramenta *OWASP ZAP*. 42
Figura 5.12 – Varredura executada pela ferramenta *Nmap*. 43
Figura 5.13 – Varredura focada na porta 3306. 43
Figura 5.14 – Varredura executada pelo *Sqlmap*. 44
Figura 5.15 – Resultado da execução do *script* que exibe na tela o *cookie*. 44
Figura 5.16 – Página inicial da aplicação *MtShop*. 47
Figura 5.17 – Página inicial do *Tenant* 1. 47
Figura 5.18 – Página *Products* 48
Figura 5.19 – Página exibida ao clicar no botão *Admin*. 49
Figura 5.20 – Análise do código da página inicial do *tenant 1* da aplicação MtShop. . . . 50
Figura 5.21 – Resultado da varredura executada com o *Dirb*. 51
Figura 5.22 – Vulnerabilidade encontrada pela ferramenta *OWASP ZAP*. 52
Figura 5.23 – Varredura executada pela ferramenta *Nmap*. 52
Figura 5.24 – Resultado da execução do *script* que exibe na tela o *cookie*. 53
Figura 5.25 – Produto que deve ser excluído com a exploração da vulnerabilidade. 54

Figura 5.26 – Captura de requisição de exclusão de produto. 54
Figura 5.27 – Requisição modificada. 55
Figura 5.28 – Página de produtos do *tenant* 2. 55
Figura 5.29 – Página da aplicação *ToyExample*. 57
Figura 5.30 – Página de *login* da aplicação *ToyExample*. 58
Figura 5.31 – Resultado da varredura executada com o *Dirb*. 59
Figura 5.32 – Vulnerabilidade encontrada pela ferramenta *OWASP ZAP*. 59
Figura 5.33 – Varredura executada pela ferramenta *Nmap*. 60
Figura 5.34 – Ataque *SQL Injection*. 60
Figura 1 – Página exibida ao clicar no botão *Product Inventory*. 70

LISTA DE TABELAS

Tabela 5.1 – Quadro de ferramentas. 35
Tabela 5.2 – Quadro de experimentos. 63

SUMÁRIO

1	**INTRODUÇÃO**	7
1.1	**Contextualização**	7
1.2	**Motivação**	9
1.3	**Objetivo**	10
1.4	**Organização**	10
2	**FUNDAMENTAÇÃO TEÓRICA**	11
2.1	**SaaS e *multi-tenant***	11
2.2	**Desafios de segurança em sistemas *SaaS***	16
2.3	**Pentest**	17
2.3.1	**Tipos de testes de invasão**	17
2.3.2	**Estágios de testes de invasão**	19
2.3.3	**Técnicas de testes de invasão**	21
2.4	**Ferramentas de Pentest**	26
3	**METODOLOGIA**	28
4	**TRABALHOS RELACIONADOS**	30
5	**TESTES DE INVASÃO EM SISTEMAS *SAAS MULTI-TENANTS***	32
5.1	**Considerações iniciais**	32
5.1.1	**Objetos de teste**	33
5.1.2	**Ambiente de teste**	33
5.1.3	**Ferramentas de teste**	35
5.1.4	**Experimentos**	35
5.2	**Experimento 1 - *iCardapio***	36
5.3	**Experimento 2 - *MtShop***	46
5.4	**Experimento 3 - *ToyExample***	56
5.5	**Considerações finais**	61
6	**CONCLUSÃO E TRABALHOS FUTUROS**	64
	REFERÊNCIAS	65
	APENDICE A – Outras funcionalidades da aplicação *MtShop*	70

1 INTRODUÇÃO

1.1 Contextualização

Nas últimas décadas, organizações passaram a depender cada vez mais dos sistemas computacionais. As tecnologias são essenciais não apenas no aspecto profissional, mas em praticamente todos os aspectos da vida de uma pessoa e no cotidiano de uma organização. Segundo Kemp (2018), em 2018 cada pessoa passou cerca de 6 horas do dia no mundo digital. O Brasil está acima da média mundial, com cada habitante passando mais de 9 horas diárias no mundo digital.

Diante da dependência dos usuários e das organizações às tecnologias e o crescente avanço em mecanismos de proteção aos dados, aumentam-se também os cyberataques que visam principalmente obter dados sigilosos e tornar sistemas inoperantes. De acordo com Cybersecurity Ventures (2016), o cybercrime custará ao mundo $ 6 trilhões ao ano até 2021. Portanto, a necessidade de investimentos na segurança de informação é fundamental.

Uma das tecnologias mais promissoras é a computação em nuvem (IEEE Computer Society, 2014), um modelo que permite acesso conveniente e sob demanda a um conjunto de recursos computacionais (como redes, servidores, armazenamento, aplicações e serviços) que podem ser rapidamente fornecidos e disponibilizados com o mínimo de esforço gerencial ou interação com os provedores do serviço (National Institute of Standards and Technology (NIST), 2010). Segundo Vaquero et al. (2008), a computação em nuvem desloca a localização da infraestrutura para a rede para reduzir custos associados com o gerenciamento de recursos de software e hardware. Este modelo surgiu a partir da contribuição de tecnologias como a computação paralela, computação distribuída e tecnologias de virtualização (RU; KEUNG, 2013).

Mäkilä et al. (2010) afirmam que, com a expansão da Internet e de tecnologias como a computação em nuvem, surgiu o modelo *Software as a Service* (*SaaS*), que são aplicações que são fornecidas aos clientes gratuitamente, ou por meio de uma taxa, mas que não pertencem ao cliente, e sim ao provedor. Os autores Mäkilä et al. (2010) também destacaram o surgimento do modelo *Software as a Service* (*SaaS*), o qual pode ser utilizado pelos clientes gratuitamente ou mediante o pagamento do uso sob demanda. Logo, tais aplicações não pertencem ao cliente, o qual apenas adquire o direito de uso, e se tornaram realidade devido à expansão da internet e de tecnologias como computação em nuvem. Diferentemente do modelo tradicional de distribuição de software, o cliente não compra nem possui uma cópia licenciada do software.

A arquitetura *multi-tenant* é um mecanismo que permite que uma única instância de um *SaaS* execute na infraestrutura do provedor do serviço, e seja compartilhada por múltiplos clientes. Em contraste com modelo multi-usuário, que permite que diversos usuários utilizem uma única instância da aplicação sem a possibilidade de customização, o modelo *multi-tenant* requer que uma única instância do software permita customizações específicas para vários usuários (KWOK; NGUYEN; LAM, 2008) (KABBEDIJK et al., 2015). Neste modelo, as organizações ou conjunto de usuários são chamados de *tenants* ou *inquilinos*. Segundo Chong, Carraro e Wolter (2006a), o modelo *multi-tenant* também se diferencia do modelo multi-instância, no qual cada cliente recebe sua própria instância (virtualizada) da aplicação.

A arquitetura *multi-tenant* alia o poder e a flexibilidade da computação em nuvem com a praticidade do modelo *SaaS* para criar uma nova categoria de aplicações que tem como principais características fornecer de forma fácil, barata e eficiente, softwares que são capazes de atender não só as necessidades básicas, bem como as necessidades específicas de cada cliente. No entanto, as características que possibilitam tal flexibilidade para as aplicações *multi-tenant*, podem também, teoricamente, aumentar a vulnerabilidade e exposição dos dados da aplicação.

Segundo (PINTO et al., 2016), o crescente uso desse padrão arquitetural vem causando um aumento no descontrole do armamento dos dados e na desconfiança em relação aos ambientes da nuvem por parte dos usuários, uma vez que um distúrbio em um recurso compartilhado pode causar impacto para todos os usuários. No entanto, os problemas-chave na computação em nuvem para aplicações *multi-tenants* no que diz respeito a vulnerabilidades, segurança e potenciais falhas continuam incertos Dey, Islam e Arif (2019), Hosni (2017), Bai et al. (2011), Gao, Bai e Tsai (2011), Vashistha e Ahmed (2012), Ru, Grundy e Keung (2014).

Chong e Carraro (2006b) ressaltam que, dependendo da estratégia de gerenciamento de dados adotada em uma aplicação *multi-tenant*, os cuidados com a segurança devem ser maiores para prevenir que um *tenant* possa acessar dados de outro durante um evento inesperado de falha na aplicação ou na ocorrência de um ataque. A atenção maior na segurança de aplicações que adotam este tipo de gerenciamento de dados se deve ao fato que os dados são compartilhados entre os *tenants*, portanto, pode haver vulnerabilidades que permitam acesso indevido na ocorrência de um ataque.

Embora haja uma falta de controle e confiança nos ambientes na nuvem, a adoção de aplicações provindas desses ambientes é crescente (VASHISTHA; AHMED, 2012). A informação e os sistemas computacionais no geral são considerados, nos dias de hoje, um dos maiores

patrimônios de uma organização, e ferramentas essenciais para suas atividades. Além disso, o ambiente de negócios no qual as empresas operam atualmente está se tornando cada vez mais complexo. Empresas, privadas ou públicas, sofrem crescentes pressões forçando-as a responder rapidamente a condições que estão sempre mudando, além de terem que ser inovadoras na maneira com que operam (TURBAN et al., 2009).

Uma maneira de aumentar a confiança sobre uma aplicação é por meio de testes que garantam a segurança da mesma. Os testes de invasão são, segundo Bacudio et al. (2011), uma série de atividades realizadas para identificar e explorar vulnerabilidades de segurança em uma aplicação com o objetivo de atestar se as medidas de segurança que foram implementadas são eficazes.

Ainda segundo Bacudio et al. (2011), os testes de invasão são importantes tanto do ponto de vista operacional, quanto do ponto de vista de negócio. Os testes de invasão são importantes do ponto de vista operacional uma vez que ajudam a traçar estratégias de segurança da informação através da identificação rápida das vulnerabilidades, e permitem a eliminação proativa dos riscos. Do ponto de vista de negócio, os testes de invasão são importantes na prevenção de perda financeira na ocorrência de uma falha de segurança, e respaldando a devida diligência e conformidade aos reguladores do setor no que se refere a segurança da informação.

Neste trabalho foi, realizado um estudo e exploração de vulnerabilidades em três aplicações *multi-tenants* que divergem em relação ao nível de isolamento de dados com o objetivo de compreender os fundamentos do padrão *multi-tenancy* e identificar vulnerabilidades em sistemas que adotam essa arquitetura. Foram realizados três experimentos de teste de invasão (PENTEST) nas aplicações selecionadas e foi possível demostrar que, devido à natureza das aplicações *multi-tenant*, o comprometimento de um *tenant* ou a exploração de algumas vulnerabilidades podem causar danos à todos os *tenants*.

1.2 Motivação

Adoção de mecanismos de prevenção de acesso indevido aos dados é fundamental para evitar danos aos usuários e para estabelecer um ambiente de confiança entre usuários e provedores de serviços. Os potenciais danos decorrentes de uma falha de segurança em aplicações *multi-tenant* podem ser muito grandes no atual contexto de dependência tecnológica das organizações que adotam sistemas *multi-tenants*.

Para que tais medidas de segurança possam ser tomadas de forma adequada, é necessário que o campo da segurança da informação em sistemas *multi-tenants* seja bem explorado e compreendido. Por esse motivo um estudo e exploração de vulnerabilidades em sistemas *multi-tenants* foi realizado.

1.3 Objetivo

O objetivo deste trabalho é identificar e explorar vulnerabilidades em sistemas baseados na arquitetura *multi-tenant*.

1.4 Organização

No Capítulo 2 são apresentados alguns conceitos básicos sobre os principais temas abordados no trabalho, como modelo *SaaS*, arquitetura *multi-tenant* e testes de invasão (Pentest). No Capítulo 3 é descrito a metodologia adotada no desenvolvimento deste trabalho. No Capítulo 4 apresentam-se pesquisas e os principais trabalhos relacionados com a proposta. No Capítulo 5 são apresentados os passos para a realização dos testes, bem como os resultados destas investigações. Por fim, no Capítulo 6 são apresentados os resultados dos testes e perspectivas futuras.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 SaaS e *multi-tenant*

Software as a Service (*SaaS*) é um modelo de entrega de serviço no qual um software é oferecido como serviço. Segundo Armbrust et al. (2010), os serviços entregues pelo modelo *SaaS* são softwares hospedados na infraestrutura da nuvem e que são fornecidos aos clientes por meio da Internet.

Mäkilä et al. (2010) listam cinco características típicas do modelo *SaaS*:

1. O produto é utilizado através de um navegador *web*.
2. O produto não é feito sob medida para cada usuário.
3. O produto não inclui software que precise ser instalado no dispositivo do usuário.
4. O produto não requer integração e instalação especial.
5. O preço do produto não é baseado na utilização do *software*, mas sim por uma taxa fixa periódica.

Além das cinco características citadas acima, existem duas outras característica que diferenciam um software oferecido pelo modelo *SaaS* para um software tradicional: a mudança na propriedade do software, que faz com que neste modelo, o software pertence ao provedor do serviço, que oferece acesso remoto aos seus clientes; e o compartilhamento de recursos para vários clientes, que permite oferecer softwares para diversos clientes utilizando os mesmos recursos computacionais, como bancos de dados e mecanismos para customizações de funcionalidades e interfaces.

Paliwal (2012) afirma que para atingir eficiência de custos na entrega de uma mesma aplicação para vários usuários, aplicações *multi-tenant* são a escolha óbvia, ao invés de aplicações desenvolvidas para um único cliente.

Com o avanço da computação em nuvem e do modelo *SaaS*, novas tecnologias emergiram e conquistaram espaço no mercado de software. Tirando proveito de uma das principais características proporcionadas pela computação em nuvem e modelo *SaaS*, o compartilhamento de recursos, surgiu a arquitetura *multi-tenant* .

Multi-tenant é o padrão arquitetural que permite que vários clientes acessem uma única instância de software executada na infraestrutura do provedor de serviço (Song; Chauvel; Solberg, 2018). Segundo Kwok e Mohindra (2008), diferentemente dos sistemas multi-usuário,

os sistemas do modelo *multi-tenant* permitem que uma única instância do software seja customizada para diferentes usuários com necessidades específicas. Além disso, os sistemas *multi-tenant* também se diferenciam dos sistemas *multi-instance*, no qual cada cliente acessa sua própria instância da aplicação (KABBEDIJK et al., 2015); (Song et al., 2019); (CHONG; CARRARO; WOLTER, 2006a).

Multi-tenant ampliou o conceito de compartilhamento de recursos e diminuiu custos na entrega e manutenção de software, mas aumentou a desconfiança já existente sobre as aplicações *SaaS*. Segundo Chong, Carraro e Wolter (2006a), os dados são os bens mais importantes de qualquer negócio, e também é o coração de um *SaaS*. Portanto, a falta de confiança é o fator número um na não adoção de um *SaaS*, pois para usufruir das vantagens e benefícios de um *SaaS*, uma organização deve abrir mão de um nível de controle sobre seus próprios dados, confiando no provedor do *SaaS* para mantê-los seguros.

Bezemer e Zaidman (2010) listam três características de aplicações *multi-tenant*:

1. Possibilidade de compartilhar recursos, com o objetivo de reduzir custos Wang et al. (2008), Warfield (2007);

2. Um alto nível de configurabilidade, possibilitando que cada *tenant* customize a aplicação de acordo com suas necessidades Nitu (2009), Jansen, Houben e Brinkkemper (2010), Müller et al. (2009);

3. Uma aplicação com uma base de dados compartilhada Kwok, Nguyen e Lam (2008).

Segundo Pinto e Souza (2019), na implementação de aplicações *SaaS multi-tenants* é necessário definir uma abordagem para a identificação dos *tenatns* para que os mecanismos da *multi-tenancy* funcionem corretamente. Devido à natureza de aplicações desta arquitetura, é necessário isolar os *tenants* em quase todas as partes da aplicação por meio de filtros. Geralmente adotada-se a utilização de um "*tenant ID*" para distinguir os *tenants* e associar cada *tenant* a seus recursos. Algumas abordagens utilizadas são a utilização de *tokens*, *tenant ID* na URI (http://app.com/tenantid/...) e *tenant ID* no *hostname* (http://tenantid.app.com).

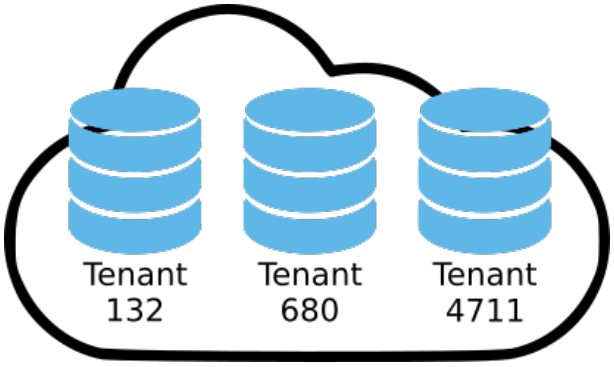
De acordo com a Salesforce (2008), *multi-tenant* só é viável quando pode sustentar aplicações confiáveis, customizáveis, aprimoráveis, seguras e rápidas. Uma forma de solucionar os problemas relacionados a segurança dos softwares *SaaS* é criando uma arquitetura de dados que atenda as necessidades de segurança dos clientes, sem abrir mão da eficiência e custos necessários para a administração por parte do provedor de serviço.

A seguir são apresentadas três abordagens propostas por Chong, Carraro e Wolter (2006a) para administrar os dados de um software *multi-tenant*.

Banco de dados isolados

Segundo Chong, Carraro e Wolter (2006a), a abordagem de administração de dados mais simples é a utilização de bases de dados separadas. Como ilustrado na Figura 2.1, nesta abordagem cada *tenant* possui sua própria base de dados que é logicamente isolada da base de dados de outros *tenants* . De acordo com Chong e Carraro (2006b), nessa abordagem uma nova base de dados padrão é criada para um novo *tenant* , e o serviço de metadados associa cada base de dados com cada *tenant*. Uma vez que a base de dados é criada, o *tenant* pode modificá-la livremente dentro do que é permitido e oferecido pela interface do usuário e lógica do software.

Figura 2.1 – Bancos de dados isolados



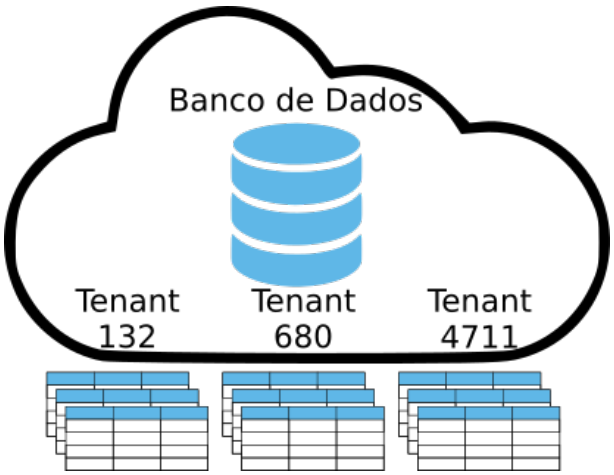
Fonte: Adaptada de (CHONG; CARRARO; WOLTER, 2006a).

Uma das principais características da arquitetura *multi-tenant* é a customização da aplicação para atender necessidades específicas de cada *tenant*. O fato das bases de dados serem isoladas nessa abordagem tornam o trabalho de customização mais simples, além de propiciar uma restauração de dados fácil e rápida em uma eventual falha no software (Song et al., 2019). Contudo, Chong, Carraro e Wolter (2006a) alegam que um software *multi-tenant* com base de dados isoladas eleva os custos de manutenção do serviço, pois os custos com hardware são maiores nessa abordagem do que em outras, já que o número de *tenants* que podem ser hospedados em um determinado servidor é limitado pelo número de base de dados que o servidor pode suportar. Para Paliwal (2012), esta abordagem é a menos adotada pelo fato de ser muito isolada e cara.

Banco de dados compartilhados com *schemas* isolados

De acordo com Chong, Carraro e Wolter (2006a), ao contrário da abordagem de banco de dados isolados, na abordagem de banco de dados compartilhados com *schemas* isolados, as bases de dados de diversos *tenants* são armazenadas em um mesmo banco, no qual cada *tenant* possui seu próprio conjunto de tabelas agrupados em um *schema* criado especificamente para o *tenant* em questão, como ilustrado na Figura 2.2. Neste modelo, as tabelas e *schemas* específicas de cada *tenant* também são criadas dinamicamente, quando cada *tenant* faz seu primeiro acesso ao sistema, e então a conta do *tenant* recebe as permissões de acesso e modificação para suas tabelas.

Figura 2.2 – Bancos de dados compartilhados, *schemas* isolados



Fonte: Adaptada de (CHONG; CARRARO; WOLTER, 2006a).

De acordo com Chong, Carraro e Wolter (2006a), as tabelas são criadas sobre uma base padrão, mas posteriormente podem ser alteradas para atender as necessidades específicas de cada *tenant*. Apesar dessa abordagem não oferecer o mesmo grau de isolamento que a abordagem de banco de dados isolados, ela oferece um isolamento lógico moderado, e suporta mais *tenants* por servidor de banco de dados do que a abordagem anterior, uma vez que não é necessário armazenar toda a estrutura de um banco de dados para cada *tenant*, mas somente a estrutura de seu *schema*. Chong, Carraro e Wolter (2006a) afirmam que uma grande desvantagem desta abordagem é a dificuldade de restaurar os dados em caso de falha, pois as atividades necessárias para restaurar os dados de um único *tenant* são complexas e podem demandar uma quantidade significativa de tempo do administrador do sistema. Segundo Paliwal (2012), esta abordagem tem adoção moderada, pois reduz os custos com recursos computacionais, mas possui alto custo de manutenção.

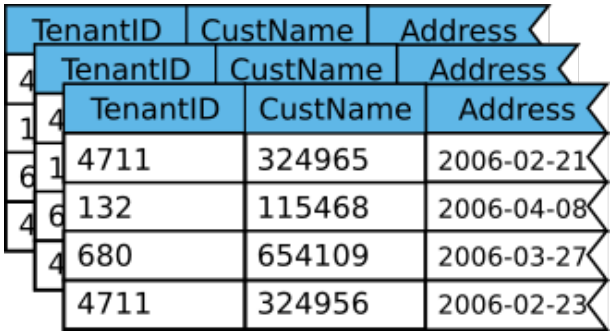
Banco de dados e *schemas* compartilhados

A terceira abordagem de administração de dados em um sistema *multi-tenant* sugerida por Chong, Carraro e Wolter (2006a) utiliza a mesma base de dados e as mesmas tabelas para diversos *tenants*. Diferentemente da abordagem anterior, na qual cada *tenant* possuía seu próprio conjunto de tabelas, na abordagem de banco de dados e *schemas* compartilhados, uma única tabela armazena dados de diversos *tenants* e contém uma coluna dedicada a associar cada registro com o *tenant* adequado, como ilustrado na Figura 2.3.

De acordo com Chong, Carraro e Wolter (2006a), essa abordagem requer menos investimentos em *hardware* e *backup*, pois permite que os dados de uma grande quantidade de *tenants* sejam armazenados em um único servidor. No entanto, esta abordagem pode demandar maior empenho e investimentos no desenvolvimento e na área de segurança, pois os dados ficam mais suscetíveis ao acesso indevido em caso de uma eventual exploração de vulnerabilidade. Apesar das ponderações referentes às áreas de desenvolvimento e segurança, Paliwal (2012) alega que esta é a abordagem mais adotada por reduzir significativamente os custos com manutenção e recursos computacionais.

Chong, Carraro e Wolter (2006a) lembram que os procedimentos para restauração de dados nessa abordagem são similares aos procedimentos da abordagem de banco de dados compartilhados com *schemas* isolados, mas enfatiza que, quando aplicado em tabelas com muitos registros, o procedimento de restauração dos dados pode causar problemas de performance para todos os *tenants* que utilizam aquela base de dados.

Figura 2.3 – Bancos de dados e *schemas* compartilhados



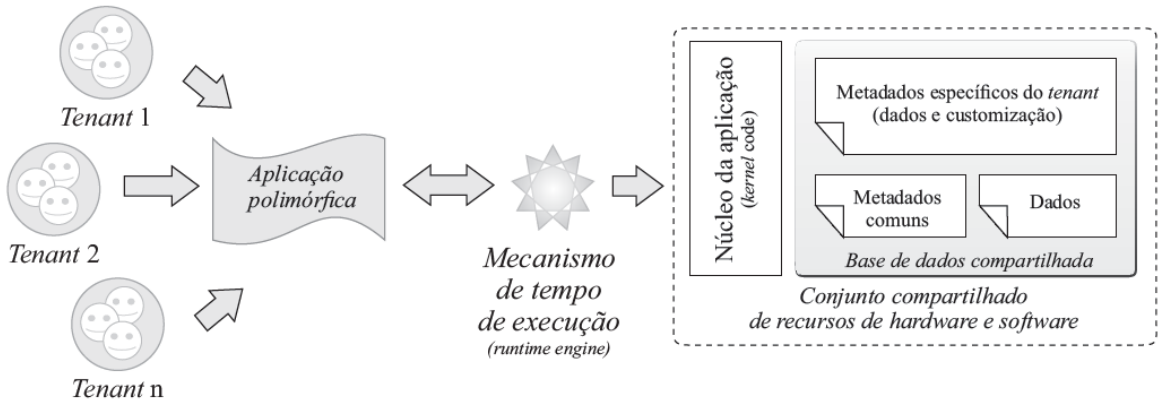
Fonte: Adaptada de (CHONG; CARRARO; WOLTER, 2006a).

Arquitetura orientada a meta-dados

Como exposto anteriormente, uma aplicação *multi-tenant* possui várias características primordiais, como a possibilidade de compartilhamento de recursos computacionais e a possibilidade de customização da aplicação por parte dos *tenants* . Independentemente da abordagem de administração de dados utilizada no desenvolvimento da aplicação, de acordo com a Salesforce (2008), tais características implicam que a mesma possui uma natureza dinâmica, polimórfica, para que seja possível atender as necessidades individuais de vários *tenants* . Por esse motivo, aplicações *multi-tenants* utilizam um mecanismo de tempo de execução (do inglês, "*runtime engine*") que gera componentes a partir de metadados Salesforce (2008).

Na arquitetura orientada a metadados, o núcleo da aplicação (*kernel*), os metadados específicos de cada *tenant* (incluindo metadados responsáveis pela customização da aplicação), os dados da aplicação e os metadados que definem as funcionalidades padrões para todos os *tenants* são claramente separados, como ilustrado na Figura 2.4 (Salesforce, 2008). Essa arquitetura permite alterações no núcleo do sistema e customizações criadas pelos *tenants* de forma independente, sem comprometer a aplicação para nenhum usuário.

Figura 2.4 – Visão geral de um sistema *SaaS multi-tenant*



Fonte: (PINTO, 2016).

2.2 Desafios de segurança em sistemas *SaaS*

Segurança é um tópico importante no desenvolvimento de qualquer software, mas as características essenciais dos sistemas *SaaS* implicam na necessidade de medidas protetivas que forneçam um alto nível de confiabilidade, uma vez que, ao contratarem o serviço, os consumidores abrem mão do controle de seus dados e essa responsabilidade passa a ser dos administradores do sistema.

De acordo com Chong, Carraro e Wolter (2006a), uma aplicação *SaaS* segura é aquela que oferece múltiplos níveis de defesa que complementam um ao outro para oferecer proteção dos dados em diferentes formas, sob diferentes circunstâncias e contra ameaças tanto internas como externas.

Chong, Carraro e Wolter (2006a) listam três padrões para oferecer os tipos certos de segurança nos lugares corretos:

- Filtragem: Utilizar uma camada intermediária entre o *tenant* e a fonte dos dados, que age como uma peneira, fazendo com que o *tenant* "visualize" a base de dados como se ela hospedasse apenas seus dados.
- Permissões: Utilizar uma lista de controle de acesso para determinar quem pode acessar os dados na aplicação e quais operações podem ser realizadas sobre os mesmos.
- Criptografia: Obscurecer os dados críticos de cada *tenant* para que se mantenham ininteligíveis em caso de acesso não autorizado.

Os padrões e abordagens descritos nos tópicos anteriores são importantes para criar uma camada de confiança e segurança em aplicações *SaaS*. Entretanto, conciliar os benefícios de sistemas desse tipo com os desafios de segurança não é uma tarefa simples. O teste de invasão (do inglês "*Penetration Test*" ou "Pentest") é um método para averiguar se as medidas de segurança tomadas durante o desenvolvimento e entrega do sistema foram suficientes. A subseção seguinte descreve os conceitos e apresenta as classificações para testes de invasão.

2.3 Pentest

Segundo Weidman (2014, p. 1), "teste de invasão" (Pentest) implica simular ataques reais para avaliar o risco associado a possíveis brechas de segurança. Um Pentest não consiste apenas em descobrir vulnerabilidades que possam ser usadas em ataques, mas também explora as vulnerabilidades para avaliar os possíveis danos de um ataque bem sucedido. Em outras palavras, um Pentest é um ataque autorizado ao sistema, que tem como objetivo encontrar e avaliar o risco potencial de vulnerabilidades.

2.3.1 Tipos de testes de invasão

Testes de invasão são classificados de acordo com a quantidade de informação fornecida previamente ao testador. Assunção (2014) define três tipos de Pentest:

Black Box (Caixa Preta)

No Pentest *Black Box*, o executor do teste possui apenas o mínimo de informação prévia necessária sobre sistema ou rede a serem testados, como nome ou endereço web da aplicação. De acordo com Assunção (2014), todos os pontos fracos e informações necessárias para um ataque bem sucedido devem ser obtidos pelo executor do teste neste modelo.

Segundo Assunção (2014), a maior vantagem deste modo de teste é simular exatamente a visão de um atacante externo sobre o sistema. Desta maneira, é possível ter certeza que todas as informações obtidas pelo testador podem ser obtidas e vazadas em uma situação real de ataque.

Assunção (2014) ressalta, no entanto, que neste modo de teste de invasão é possível obter apenas a visão de um invasor externo, porém a maioria dos ataques ocorrem no ambiente interno das organizações. Além disso, o tempo necessário para executar um teste de invasão no modo *Black Box* é maior do que nos outros dois modos.

White Box (Caixa Branca)

Segundo Bacudio et al. (2011), no modo de teste de penetração *White Box*, o testador é munido com todas as informações necessárias para testar o alvo. Assunção (2014) afirma que os objetivos do Pentest *White Box* são mais específicos, e que este tipo de teste normalmente é utilizado para descobrir vulnerabilidades no sistema, e não em ter a perspectiva do invasor na execução de um ataque ou averiguar vazamentos de informações, como no modo *Black Box*.

Uma grande vantagem o modo *White Box* é que o tempo necessário para executar o teste é significativamente inferior ao tempo necessário para executar um teste *Black Box*.

Grey Box (Caixa Cinza)

No modo *Grey Box*, o testador é munido de conhecimento parcial a respeito da rede e do sistema a serem testados. Segundo Assunção (2014), este teste é o mais utilizado para simular ataques realizados por alguém de dentro da organização. Este modo é mais adequado nessa situação, pois normalmente um funcionário possui algumas informações sobre a rede e sistemas com os quais tem contato, o que o provê mais informações iniciais que um testador possuiria em um teste *Black Box*, e menos do que um testador possuiria em um teste *White Box*.

2.3.2 Estágios de testes de invasão

Um teste de invasão é um processo complexo, longo e metódico, por isso é comum no meio acadêmico que se divida o teste em estágios, no entanto, não há consenso quanto a quantidade e propósito dos estágios. Alguns autores, como Gregg (2006), dividem o Pentest em seis estágios: reconhecimento, escaneamento e enumeração, ganhando acesso, escalonamento de privilégios, mantendo Acesso e cobrindo rastros. Outros autores segmentam o teste de invasão em menos estágios, como Assunção (2014), que divide em três fases: planejamento, execução do teste e fase pós-teste.

Para o contexto deste trabalho, adota-se o desmembramento proposto por Weidman (2014), por ser uma divisão com segmentos mais coesos. Nessa proposta, o teste de invasão é dividido em sete estágios:

Pré-engajamento

Pré-engajamento é o estágio que precede o início do Pentest em si. De acordo com Weidman (2014), nesse estágio o testador deve-se certificar de que ele e o contratante do Pentest estão em harmonia em relação ao procedimento. Perguntas relacionadas às maiores preocupações do contratante, fragilidade de sistemas e dispositivos interligados e quais os principais objetivos do Pentest devem ser feitas nessa etapa.

Weidman (2014) afirma que outros itens importantes que devem ser discutidos e pactuados são:

- **Escopo**: Quais sistemas e redes devem ser testadas e quais não? O testador pode utilizar ferramentas e técnicas que podem derrubar o serviço? O contratante está ciente de que os procedimentos podem causar instabilidade no sistema? O testador pode realizar testes de ataques que utilizam engenharia social?
- **Período de teste**: Em qual período os testes podem ser realizados?
- **Informações de contato**: Quem deve ser contatado caso algo significativo seja encontrado? O contato pode ser feito a qualquer hora? O contato deve ser feito por um meio protegido (*e-mail* criptografado, por exemplo)?
- **Autorização**: Certificar-se de que possui autorização por escrito para realizar os procedimentos.

- **Termos de pagamento**: Como, por quem, quando e de que forma será realizado o pagamento?

Além dos cinco itens apresentados acima, Weidman (2014) ressalta a importância da inclusão de um acordo de confidencialidade no contrato.

Coleta de informações

Nesta etapa o testador explora os sistemas e redes alvos livremente à procura de fontes de informações. Weidman (2014) afirma que nessa fase se dá o início de utilização de ferramentas, como *port scanners*, que tem como objetivo identificar *hosts* na rede, encontrar portas abertas, detectar versões de sistemas, entre outros.

Modelagem de ameaças

As informações coletadas na etapa anterior servem de base para o desenvolvimento dos planos e estratégias de ataque a serem elaborados. Por exemplo, se o cliente realiza empréstimos financeiros, um atacante poderia arruinar a organização se conseguisse modificar informações armazenadas a respeito dos empréstimos realizados.

Análise de vulnerabilidades

Após realizar a modelagem de ameaças, o testador começa a procurar por vulnerabilidades para determinar se suas estratégias de invasão são viáveis. Apesar de ferramentas como *vulnerability scanners* serem muito eficazes e utilizadas nesta etapa, elas não substituem o pensamento crítico, portanto a análise manual também deve ser efetuada pelo testador.

Exploração

As vulnerabilidades analisadas na etapa anterior são alvo de ferramentas e técnicas com o objetivo de conseguir acesso ao sistema e suas informações. O sucesso nesta etapa significa que o testador, e um potencial atacante, são capazes de invadir o sistema alvo, mas até onde o intruso consegue chegar é determinado na próxima etapa.

Pós-exploração

Nessa etapa o testador já possui acesso ao sistema, mas o risco potencial na exploração desta vulnerabilidade ainda é incerto. O nível do risco é mensurado de acordo com o que é possível obter a partir do momento da invasão em relação aos seguintes itens: informações do sistema; análise de arquivos; tentativa de aumento de privilégios; e descoberta de mais vulnerabilidades que podem conceder acesso a outros sistemas.

Relatórios

Na última etapa o testador deve relatar suas descobertas e conclusões. O testador deve apresentar suas descobertas ao cliente de forma objetiva. De acordo com Weidman (2014), no relatório deve ser apontado o que está sendo feito corretamente e o que deve ser aprimorado em relação às medidas de segurança. Além disso, deve ser relatado como o testador conseguiu acesso, o que foi encontrado e como resolver o problema.

2.3.3 Técnicas de testes de invasão

Atualmente, existem diversos tipos de testes de invasão que podem ser classificados em várias categorias. As categorias de testes de invasão podem diferenciar-se quanto ao tipo de alvo, ambiente de aplicação ou objetivo do teste. Vários tipos de testes de invasão não são adequados para o objetivo do trabalho e portanto, apenas testes de invasão utilizados nos estudos experimentais do trabalho serão explorados nesta sessão.

Varredura (*Scan*)

A varredura não é um ataque de exploração de vulnerabilidade, mas é uma forma de ataque e, de acordo com a CERT (2018), foi o tipo de ataque mais comum em domínios *.br* em 2018. Os ataques do tipo *scan* normalmente são executados na etapa de *Coleta de informações* e são um dos primeiros testes de invasão realizados por um testador durante um Pentest, pois é através destes testes que é possível coletar grande parte das informações necessárias para a realização de outros tipos de ataques.

Segundo Assunção (2014), as varreduras são fases muito importantes de um *Pentest*, pois seus resultados podem indicar o sucesso ou fracasso do processo. As varreduras não têm o propósito de obter acesso a dados protegidos, danificar ou modificar sistemas, mas sim de obter

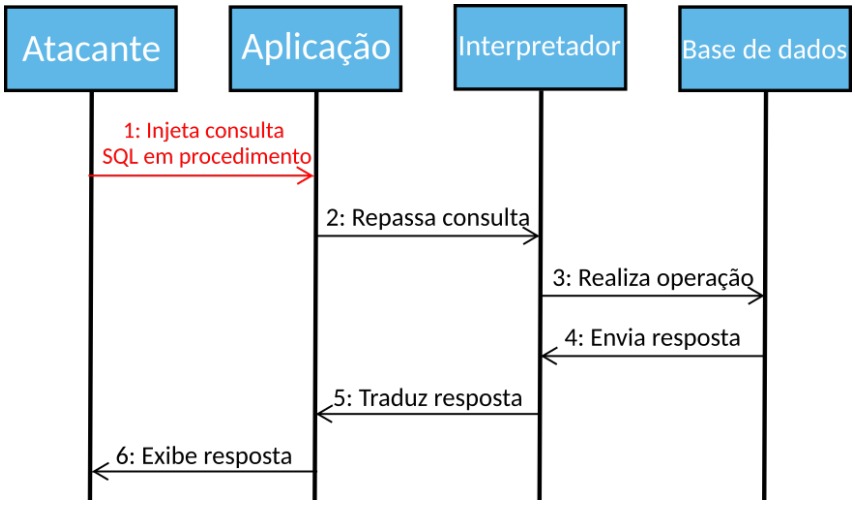
informações úteis para outros ataques, como portas abertas que podem indicar a execução de algum serviço com vulnerabilidae conhecida. De acordo com Assunção (2014), uma varredura consiste em fazer requisições aos servidores do sistema alvo em busca de identificar computadores na rede, portas abertas nos computadores, sistema operacional, páginas não indexadas, serviços ativos, usuários, falhas locais e falhas remotas.

SQL Injection

De acordo com a comunidade online OWASP (2017), que cria e disponibiliza artigos, metodologias, documentação, ferramentas e tecnologias no campo da segurança da informação, *SQL Injection* foi a vulnerabilidade mais crítica em 2017. Uma falha de *SQL Injection* ocorre quando o interpretador de *SQL* é confundido por meio de comando enviado a partir de uma entrada de dados de usuário como parte de um comando ou consulta. Desta forma um atacante pode ter acesso indevido aos dados da aplicação. Segundo Basso (2010), aplicações se tornam vulneráveis a ataques de injeção de *SQL* quando entradas de usuários fornecidas a um interpretador não são validadas ou re-codificadas.

A Figura 2.5 ilustra a sequência de como geralmente ocorre um ataque do tipo *SQL Injection*. Note que, em suma, o que é representado na figura é uma realização de operação *SQL* comum. Isso ocorre porque um ataque *SQL Injection* não modifica a forma como as operações em banco de dados são realizadas, apenas insere dados em procedimentos já existentes da aplicação de forma que o interpretador da linguagem processará a entrada de dados como parte da operação *SQL*.

Figura 2.5 – Sequência de um ataque *SQL Injection*



Fonte: Elaborada pelo autor.

Exemplo de ataque *SQL Injection*

Considere o cenário onde a aplicação possua um campo de login no sistema construído com uma consulta *SQL* da seguinte forma:

```
SELECT * FROM users WHERE username = campo_login AND senha = campo_senha
```

Neste cenário, se um atacante preencher os campos login e senha com o comando:

```
'or'1'='1
```

e enviar a requisição para o servidor, o interpretador da linguagem *SQL* se confundirá e, internamente, o comando *SQL* resultará na seguinte instrução:

```
SELECT * FROM users WHERE username = 'or'1'='1 AND senha = 'or'1'='1
```

Note que com o comando inserido no campo de senha, independente do login e senha informados, a condição será sempre verdadeira, permitindo o acesso do atacante à aplicação sem possuir a devida permissão.

Por meio da exploração dessa vulnerabilidade o atacante pode acessar, adicionar, excluir e modificar qualquer informação armazenada no banco de dados. No cenário em que o sistema alvo é um sistema *multi-tenant*, o atacante teria o poder de realizar qualquer uma das ações citadas acima sobre os dados de qualquer *tenant* daquela aplicação.

***Cross-site Scripting* (*XSS*)**

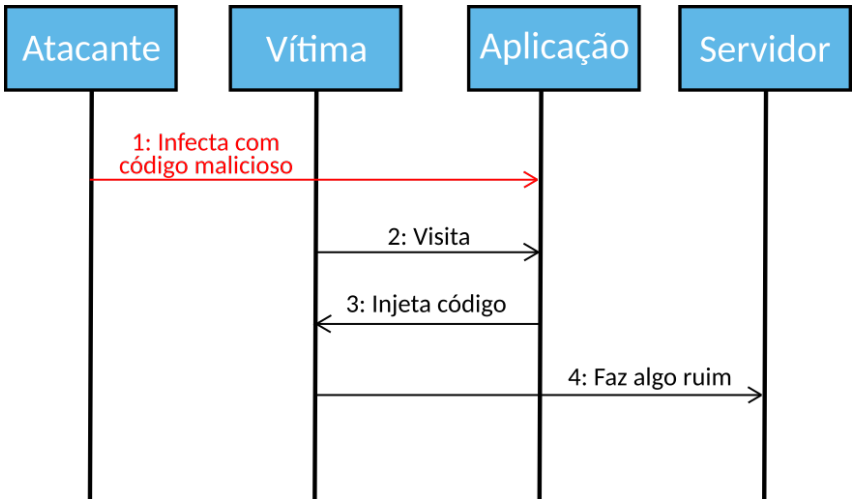
De acordo com Torres (2014), um ataque do tipo *XSS* consiste na inserção de um código que é capaz de explorar o interpretador do navegador com objetivo de roubar informações, roubar sessões, redirecionar para outros sites ou inserir código malicioso na página. A Figura 2.6 ilustra uma sequência como ocorre um ataque do tipo *XSS*.

Silva et al. (2015) alegam que existe dois tipos de ataques *Cross-site Scripting*: Refletido (*Reflected*) e Armazenado (*Stored*).

Refletido (*Reflected*)

Segundo Torres (2014), o tipo *reflected* é o mais comum, mas também é considerado o tipo que oferece menor risco. Neste tipo de ataque, o código malicioso é executado pelo

Figura 2.6 – Sequência de um ataque *XSS*



Fonte: Adaptada de (BILBAO, 2014).

navegador da vítima, mas não é armazenado nem no computador da vítima, nem no servidor da aplicação.

Armazenado (*Stored*)

O tipo *Stored* é considerado o mais perigoso pois, nesse tipo de ataque, é possível armazenar o código malicioso na aplicação, tornando o ataque presumivelmente maior em relação a quantidade de usuários afetados, uma vez que, a partir do momento da injeção do código malicioso, qualquer usuário que acessar a aplicação estará suscetível ao ataque. Alvos comuns deste tipo de ataque são aplicações que possuem campos nos quais é possível inserir comentários como fóruns, blogs e sites de notícia.

Exemplo de ataque *Cross-site Scripting* (*XSS*)

Considere o cenário onde a aplicação possua um campo no qual o usuário pode inserir alguma informação, e essa informação será exibida na tela. Considere que o campo em questão seja uma área para inserção de comentários. Se o código responsável pela exibição dos comentários simplesmente copia a informação inserida pelo usuário e a insere no código na página *HTML* sem qualquer verificação, qualquer código inserido neste campo será executado pelo interpretador da linguagem.

Suponha que o seguinte código seja inserido no campo de comentários:

```
<script>
var elements = document.getElementsByClassName(container);
while(elements.length > 0){
elements[0].parentNode.removeChild(elements[0]);
}
</script>
```

Como não há nenhuma verificação dos dados inseridos, a aplicação irá inserir o código acima na página *HTML*, fazendo com que o interpretador de *JavaScript* execute o código sempre que a página com os cometários for acessada. O código acima excluirá todo o conteúdo contido na *div container*.

Dessa forma, não haveria exposição dos dados dos *tenants*, porém outros códigos poderiam ser utilizados na exploração desta vulnerabilidade, caracterizando outros tipos de técnicas de invasão, como o *Cross-site Request Forgery*.

Cross-site Request Forgery (*CSRF*)

CSRF pode ser considerado um subtipo de *XSS*, uma vez que ambos utilizam as mesmas vulnerabilidades e princípios. Entretanto, no *CSRF* o invasor rouba as credenciais de um usuário com sessão ativa, na qual a aplicação confia, para utilizá-las em ações não autorizadas.

Suponha que o seguinte código seja inserido no campo de comentários:

```
<script>window.location=http://www.atacante.com/?=+document.cookie
    ↪ </script>
```

Como não há nenhuma verificação dos dados inseridos, a aplicação irá inserir o código acima na página *HTML*, fazendo com que o interpretador de *JavaScript* execute o código sempre que a página com os cometários for acessada. O código acima enviará o *cookie* do usuário para o servidor *www.atacante.com*, que estará sendo monitorado pelo atacante. O atacante poderá, então, utilizar o *cookie* enviado para conseguir acesso não autorizado ao sistema. Este ataque é conhecido como *roubo de sessão*.

Em um cenário em que o *cookie* enviado foi de um usuário administrador, e que o sistema alvo é um sistema *multi-tenant*, todos os *tenants* poderiam ser afetados por essa vulnerabilidade.

2.4 Ferramentas de Pentest

Para detectar características e falhas presentes em aplicações web, existem no mercado centenas de ferramentas dedicadas a áreas específicas. As ferramentas selecionadas para serem utilizadas nesta pesquisa foram ferramentas preferencialmente *open source*, gratuitas e disponíveis para o sistema operacional Linux. Outro fator levado em consideração na escolha das ferramentas foi o *ranking* SecTools (2019), *site* de segurança da informação mantido por Gordon Lyon, especialista em segurança de rede autoproclamado *hacker* "Fyodor"que lista as cento e vinte cinco ferramentas de segurança mais utilizadas e reconhecidas pela comunidade de segurança da informação. O *ranking* se baseia na opinião de milhares de profissionais da área que concedem notas às ferramentas.

Nmap

O *Nmap*[1] é uma ferramenta de varredura *open source* desenvolvida por Gordon Lyon, especialista em segurança de rede autoproclamado hacker "Fyodor". O *Nmap* é, segundo Assunção (2014), a ferramenta de varredura mais popular atualmente. É considerada por Weidman (2014) um padrão da indústria. É extremamente poderosa, contendo diversos recursos, e possui vários livros dedicados a ela.

Alguns dos recursos do *Nmap* são: descoberta de *host*, varredura de portas, detecção de serviços e detecção de sistema operacional.

Sqlmap

O *Sqlmap*[2] é uma ferramenta de testes de invasão automatizada *open source* que, segundo Torres (2014), automatiza o processo de detecção e exploração de falhas de *SQL Injection*. O propósito da utilização desta ferramenta é conseguir acesso a banco de dados de aplicações web vulneráveis.

Entre vários recursos do *Sqlmap*, destacam-se:

- Suporte a diversos tipos de *SGBD*, como *MySQL*, *Oracle*, *PostgreSQL*, *Microsoft SQL Server*, *SQLite*, *Firebird* e outros.
- Suporte a seis diferentes técnicas de *SQL Injection*.

[1] https://nmap.org/
[2] http://sqlmap.org/

- Suporte a enumeração de usuários, senhas armazenadas em *hash*, privilégios, tabelas, colunas.
- Suporte a pesquisa de nome de banco de dados, tabelas ou colunas.
- Suporte a *download* e *upload* de qualquer arquivo do servidor do banco de dados quando o *SGBD* for *MySQL*, *PostgreSQL* ou *Microsoft SQL Server*.

Burp Suite

Burp Suite[3] é uma ferramenta de *Pentest* desenvolvida pela *PostWigger*. É muito utilizada no mercado de segurança por auditores, pesquisadores e *pentesters* por conter vários módulos que podem ser utilizados em vários tipos de aplicações. A ferramenta é dividida nos seguintes componentes:

- *Burp Proxy*: Inspeciona e modifica o tráfego entre o navegador e a aplicação destino.
- *Burp Spider*: Realiza rastreamento de conteúdo dentro de aplicações *web*.
- *Burp Scanner*: Detecta vários tipos de vulnerabilidades.
- *Burp Intruder*: Explora vulnerabilidades.
- *Burp* Repetidor: Manipula e reenvia pedidos entre o navegador e a aplicação destino.
- *Burp* Sequenciador: Testa a aleatoriedade de *tokens* de sessão.
- *Burp* Decodificador: Reconhece e decodifica vários formatos de codificação.
- *Burp* Comparador: Compara dados da aplicação.

OWASP ZAP

O *OWASP Zed Attack Proxy Project*[4] (ou *OWASP ZAP*) é uma das mais populares ferramentas gratuitas de varredura *open source*. Mantida pela fundação *OWASP*, a ferramenta possui características que a destacam dentre as ferramentas da área, como interface gráfica, multiplataforma e tradução para mais de 20 idiomas.

[3] https://portswigger.net/burp
[4] https://owasp.org/index.php/OWASP$_{Z}ed_{A}ttack_{P}roxy_{P}ro\,ject$

3 METODOLOGIA

Nesta seção descreve-se a metodologia de pesquisa que foi utilizada para realização deste trabalho.

De acordo com Jung (2004), a metodologia de pesquisa se caracteriza pelo conjunto de métodos, técnicas e procedimentos com o objetivo tornar principal de viabilizar a execução da pesquisa, que por sua vez resulta em um novo produto, processo ou conhecimento. Abaixo serão descritos e classificados o tipo de pesquisa, procedimentos metodológicos e abordagens realizadas neste trabalho.

Tipo de Pesquisa

Segundo Silva e Menezes (2005), existem várias maneiras de se classificar uma pesquisa, sendo os pontos mais tradicionais: quanto à natureza da pesquisa, a forma de abordagem do problema, os seus objetivos e os procedimentos técnicos.

Em relação à natureza, este trabalho pode ser classificado como básica, uma vez que o objetivo é gerar conhecimentos para o avanço da ciência sem aplicação prática prevista.

Quanto aos objetivos da pesquisa, este trabalho pode ser considerado como exploratório uma vez que visa identificar e explorar vulnerabilidades de um determinado tipo de aplicação.

Com relação à abordagem do problema, a classificação da pesquisa é considerada como qualitativa pois, fenômenos serão interpretados e atribuídos significados como parte da análise dos resultados dos experimentos, sem utilização de métricas e técnicas estatísticas.

Quanto aos procedimentos, este trabalho pode ser considerado como estudo de caso múltiplo, pois envolve o estudo de aplicações sob a ótica do assunto definido para este trabalho.

Os métodos para coleta de dados podem ser classificados como experimentação, pois se realizará vários testes a fim de obter dados para a análise final.

Os passos para consecução do objetivo geral deste trabalho são 3:

- Estudar a arquitetura *multi-tenant*: Neste passo, o aluno deverá ler e aprender sobre a arquitetura e os fundamentos que caracterizam uma aplicação *multi-tenant*.

- Estudar vulnerabilidades: Neste passo, o aluno deverá aprender a quais vulnerabilidades a arquitetura *multi-tenant* está suscetível de forma a poder testar as aplicações e verificar se elas possuem tais vulnerabilidades.

- Conduzir experimentos: Neste passo, o aluno deverá executar testes de invasão para identificar vulnerabilidades e avaliar o nível de exposição dos dados das aplicações testadas.

4 TRABALHOS RELACIONADOS

Hawedi, Talhi e Boucheneb (2018) alegam que atender os requisitos de segurança dos *tenants* ainda é um grande desafio para os provedores de serviços na nuvem. Os autores afirmam que aplicações que utilizam a computação em nuvem estão significativamente mais expostas a tentativas de invasão, e que os mecanismos de defesa fornecidos oferecem o mesmo grau de segurança para todos os *tenants*, não havendo a possibilidade de personalização. Em seu trabalho, propuseram um sistema de detecção de invasão *mutli-tenant* (do inglês *multi-tenant intrusion detection system* ou *MTIDS*) com o objetivo de reduzir custos e oferecer um mecanismo de defesa robusto.

Para justificar sua proposta, Hawedi, Talhi e Boucheneb (2018) conduziram alguns experimentos no qual submeteram o *MTIDS* a diversos cenários. Os resultados mostram que o *framework* possui um maior custo benefício quando comparado com *frameworks* de abordagens existentes. A abordagem proposta permite que o provedor do serviço de segurança maximize seus ganhos e utilize seus recursos com mais eficiência. Além disso, os resultados também demonstraram que utilizar um sistema de detecção de invasão com todas as funcionalidades, aumenta o consumo de recursos ao ponto de que seja necessário a inicialização de mais máquinas virtuais, aumentando o custo para o cliente. Com a abordagem proposta pelos autores, é possível utilizar o sistema de detecção de invasão com um número limitado de recursos, podendo eliminar a necessidade de utilização de mais recursos da nuvem.

Ao contrário da proposta de Hawedi, Talhi e Boucheneb (2018), este trabalho de conclusão de curso não almejou propor um novo *framework*, ferramenta ou técnica para contribuir com o desenvolvimento da segurança na nuvem. O objetivo deste trabalho foi compreender os mecanismo principais da arquitetura *multi-tenant* e conduzir estudos explorando vulnerabilidades em sistemas *multi-tenants* com apoio de testes de invasão.

Zissis e Lekkas (2012) alegam que a rápida transição de mainframes e modelos cliente/servidor para a nuvem alimentou preocupações sobre questões críticas para o sucesso dos sistemas de informação, comunicação e segurança da informação. Os autores afirmam que pela perspectiva de segurança, um grande número de riscos e desafios desconhecidos foram introduzidos por essa realocação para as nuvens, fazendo com que muitos mecanismos de proteção tradicionais perdessem sua eficácia. Os autores propuseram avaliar a segurança na nuvem por meio da identificação de requisitos de segurança únicos neste ambiente e procuraram apresentar uma solução viável que eliminasse potenciais ameaças.

A proposta dos autores de uma solução que elimina as potenciais ameaças para aplicações na nuvem é uma combinação de políticas, protocolos e mecanismos que busca garantir a autenticação, integridade e confidencialidade dos dados e comunicações dessas aplicações.

O trabalho de Zissis e Lekkas (2012) expõe como a transição para a computação na nuvem criou novos problemas de segurança para os sistemas de informação e propôs uma solução para mitigar essas ameaças, contudo não trata da avaliação da exposição dos dados de aplicações hospedadas nesse modelo.

LaBarge e McGuire (2013) realizaram uma pesquisa exploratória sobre softwares na nuvem por meio da realização de vários testes de invasão no *software OpenStack Essex Cloud Management*. Os autores apresentam alguns tipos de testes de invasão que podem ser utilizados em aplicações na nuvem, assim como diferentes ferramentas que podem ser utilizadas nesses testes. Alguns testes foram bem sucedidos e foi possível descobrir e explorar algumas vulnerabilidades como roubo de credenciais (*credential theft*), na qual foi possível descobrir as credenciais de login de administrador. Os autores afirmam ainda que todas as vulnerabilidades encontradas e exploradas no trabalho poderiam ser eliminadas com a utilização de criptografia. O roubo de credenciais poderia ser evitado com a utilização de HTTPS no lugar de HTTP para comunicações entre os usuários e o provedor da nuvem.

Assim como neste trabalho, os testes de invasão efetuados por LaBarge e McGuire (2013) tiveram como alvo aplicações que podem ser hospedadas na nuvem. Entretanto, as aplicações testadas são aplicações convencionais, e não aplicações *multi-tenant*. Contudo, a metodologia adotada pelos autores para descoberta e exploração de vulnerabilidades podem ser adaptadas para testes em aplicações *multi-tenants*.

5 TESTES DE INVASÃO EM SISTEMAS *SAAS MULTI-TENANTS*

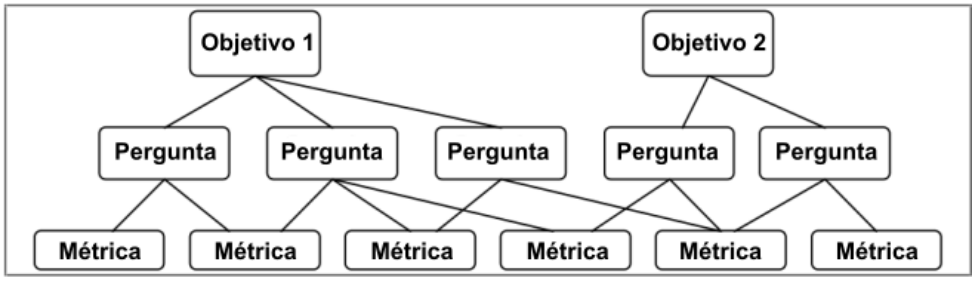
5.1 Considerações iniciais

Para a realização dos testes de invasão dos experimentos, foi adotado o modelo de medição GQM (do inglês *Goal Question Metric*) (BASILI; WEISS, 1984). GQM segue uma abordagem *top-down* na qual primeiro definem os objetivos, formula-se perguntas e, então, determina-se métricas.

A estrutura do *GQM* é apresentada na Figura 5.1. O modelo GQM possui três níveis:

- Objetivos/metas: Neste nível determina-se metas/objetivos para um determinado objeto, em vista de uma certa finalidade, em relação a uma característica, segundo um ponto de vista e em um contexto específico.
- Perguntas/questões: Neste nível elabora-se perguntas para que, ao serem respondidas, conclua-se se a meta/objetivo foi alcançada.
- Métricas/medidas: Neste nível determina-se como as perguntas podem ser respondidas.

Figura 5.1 – Estrutura do GQM.



Fonte: Adaptado de Basili, Caldiera e Rombach (1994).

Apesar de o método GQM possuir três níveis, Basili e Weiss (1984) estabeleceram seis etapas básicas:

- Estabelecer um conjunto de metas/objetivos.
- Desenvolver um conjunto de perguntas/questões.
- Determinar métricas/medidas.
- Estabelecer mecanismos para coleta de dados.
- Coletar e validar os dados.

- Analisar dados.

Para atingir o objetivo de compreender os mecanismos principais da arquitetura *multi-tenant* e conduzir estudos explorando vulnerabilidades em sistemas *multi-tenants* com apoio de testes de invasão, seguiu-se as seis etapas do método GQM definidas por Basili e Weiss (1984) em conjunto com os sete estágios de *Pentest* definidos por Weidman (2014), conforme descrito anteriormente no Capítulo 2.

5.1.1 Objetos de teste

Para a realização dos testes de invasão, utilizou-se três aplicações *multi-tenant*:

- *iCardapio*: é uma aplicação para criação de cardápios virtuais. Ela permite que cada *tenant* crie uma página e a personalize com seus dados e informações sobre os produtos vendidos pelo estabelecimento (MANDUCA et al., 2014).

- *MtShop*: é uma aplicação que permite a criação de lojas virtuais nas quais é possível cadastrar produtos para venda. Ao contrário do *iCardapio*, onde é possível apenas exibir os produtos que são vendidos no estabelecimento, o *MtShop* possibilita a venda *online* dos produtos (PINTO; SOUZA; SOUZA, 2019a).

- *ToyExample*: aplicação *mutl-tenant* desenvolvida posteriormente aos testes para que vulnerabilidades não encontradas no *iCardapio* e *MtShop* fossem exploradas. Pelo fato dessa aplicação ter sido criada apenas para a realização de experimentos, ela possui poucas funcionalidades, apenas uma tela de *login* e uma página com uma mensagem informando o *tenant* referente a cada instância.

As aplicações *iCardapio* e *Mtshop* foram selecionadas para os experimentos por serem aplicações *open source* e por adotarem diferentes abordagens de persistência dos dados. Entretanto, não foi possível explorar vulnerabilidades de *SQL Injection* para avaliar o nível de exposição de dados nessas aplicações. Por este motivo, e devido a dificuldade de encontrar outras aplicações que poderiam ser utilizadas nos experimentos, foi desenvolvida uma aplicação *multi-tenant* com base em práticas recentes de desenvolvimento.

5.1.2 Ambiente de teste

Para a realização dos testes de invasão empregados neste trabalho, as aplicações-alvos foram hospedadas na plataforma *Heroku*. Segundo Solórzano e Charão (2017), o *Heroku* é

uma plataforma que disponibiliza um ambiente de computação na nuvem com suporte a diversas linguagens de programação. As aplicações executadas na plataforma são executadas em contêineres *Unix* isolados e virtualizados.

O tipo dos testes de invasão empregados nos dois primeiros experimentos deste trabalho pode ser caracterizado como *Grey Box*, dado que o testador teve a sua disposição os trabalhos de Manduca et al. (2014) e Pinto, Souza e Souza (2019a), nos quais as aplicações *iCardapio* e *MtShop* são apresentadas. O teste empregado no terceiro experimento (aplicação *ToyExample*, pode ser caracterizado como *White Box*, pois todas as informações sobre a aplicação eram de conhecimentos do testador.

O *iCardapio*[1] foi escolhido por ser *open-source* e atender múltiplos *tenants* a partir de uma única instância. Assim, as versões podem ser definidas e configuradas para execuções simultâneas dos testes. De acordo com os autores Manduca et al. (2014), essa aplicação foi desenvolvida com as seguintes tecnologias: *Spring*[2], *Spring Tool Suite*[3], *MySQL*[4], *Maven*[5], *EclipseLink*[6] e *Java*. Destaca-se que o código fonte está disponível[7] e foi projetado para permitir qualquer provedor *SaaS* realizar o download e utilizá-lo diretamente.

O *MtShop* foi desenvolvido utilizando as seguintes tecnologias: *Spring Boot*, *AngularJS*[8], *Bootstrap*[9], *jQuery*[10], *Cloudinary*[11] para gerenciamento de imagens na nuvem, *JPA*, *MySQL*, *Maven*, *Hibernate*[12] e *Java*, como linguagem de implementação. Quanto a persistência de dados, banco de dados isolados são utilizados em cada instância da aplicação designada para cada *tenant*.

A aplicação *ToyExample* foi desenvolvida com a linguagem *Java* e utiliza as tecnologias: *MySQL*, *Hibernate*, *Spring Boot*, *JPA*, *Maven* e *Java*. A abordagem de banco de dados isolados foi utilizada na persistência de dados. O código da aplicação está disponível[13] para download e pode ser implantado diretamente no *Heroku*.

[1] https://icardapio.herokuapp.com/
[2] https://spring.io.
[3] https://spring.io/tools/sts
[4] https://mysql.com
[5] https://maven.apache.org/
[6] https://eclipse.org/eclipselink/
[7] https://github.com/michetti/icardapio/tree/multitenant
[8] https://angularjs.org/
[9] https://getbootstrap.com/
[10] https://jquery.com/
[11] https://cloudinary.com
[12] https://hibernate.org/
[13] https://drive.google.com/open?id=1P5jbI8qzZwMsvw6XlzqlcVsm0o0M8hTi

5.1.3 Ferramentas de teste

Várias ferramentas foram utilizadas na realização dos testes de invasão. A Tabela 5.1 sumariza as ferramentas utilizadas nesse trabalho.

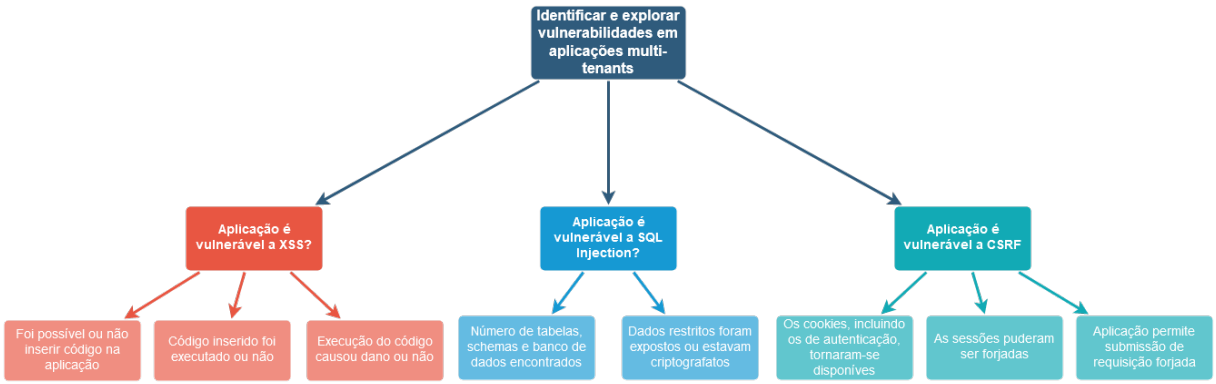
Tabela 5.1 – Quadro de ferramentas.

Ferramenta	Estágio	Descrição
Dirb	Coleta de informações	Scanner
OWASP ZAP	Análise de vulnerabilidades	Scanner
Nmap	Análise de vulnerabilidades	Scanner
Burp Suit	Exploração	Scanner
Sqlmap	Exploração	Exploit

5.1.4 Experimentos

Após definir a metodologia, objetos de teste, ambiente de teste e ferramentas, as três primeiras etapas do GQM foram executadas. O objetivo, questões e as respectivas métricas são apresentadas na Figura 5.2.

Figura 5.2 – Diagrama GQM do trabalho.



Fonte: Elaborado pelo autor.

As etapas quatro e cinco do método GQM têm por objetivo estabelecer mecanismos para coleta e validação dos dados. O mecanismo definido para a coleta dos dados neste trabalho está contido nas fases iniciais do *Pentest*. Vários testes de invasão foram aplicados nos objetos de teste seguindo os estágios propostos por Weidman (2014). Os testes são apresentados nas subseções seguintes.

5.2 Experimento 1 - *iCardapio*

Estágio 1. Pré-engajamento

Em um cenário no qual uma organização ou indivíduo contrataria um profissional para realizar um teste de invasão em sua aplicação, nessa etapa seriam realizados discussões e acordos em relação as dúvidas e preocupações sobre o teste. Entretanto, por se tratar de experimentos em instâncias de aplicações dedicadas a esse fim, tais discussões e acordos foram realizados apenas entre os envolvidos no trabalho. As orientações dadas ao aplicador dos testes foram acerca do endereço web da aplicação, o fato de que esta instância do sistema existia apenas para este fim e o usuário e senha padrão do *tenant*.

Estágio 2. Coleta de informações

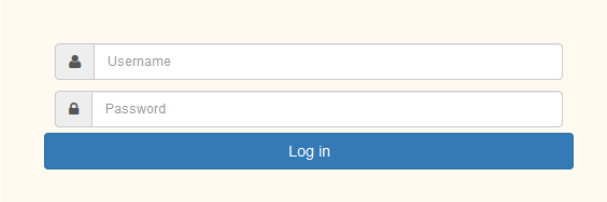
A coleta de informações iniciou-se com o reconhecimento da aplicação por meio do acesso e análise da página principal. Na Figura 5.3 ilustra-se a página inicial da aplicação, na qual é possível criar um novo cardápio eletrônico (*tenant*).

Figura 5.3 – Página inicial da aplicação *iCardapio*.

Fonte: Captura de tela realizada pelo autor.

Após o cadastro de um novo *tenant*, o cliente é direcionado para a página do seu estabelecimento, na qual são exibidas as informações do estabelecimento, produtos cadastrados no cardápio e o botão *Entrar*. Ao acionar o botão *Entrar*, o *tenant* é direcionado para a página exibida na Figura 5.4, na qual é possível inserir o nome de *tenant* e senha para acesso ao painel administrativo da página do *tenant*.

Figura 5.4 – Página inicial da aplicação *iCardapio*.



Fonte: Captura de tela realizada pelo autor.

Após iniciar a sessão, o botão *Entrar* é substituído pelo botão *Ação* que possui duas opções: *Adicionar Produto* e *Sair*. Ao clicar em *Adicionar Produto*, um modal é exibido com campos para inserção de dados sobre um produto a ser adicionado ao cardápio, como exibido na Figura 5.5. Preencher as informações sobre o produto e clicar no botão Adicionar adiciona o produto ao cardápio, exibindo-o na página do *tenant*, como mostrado na Figura 5.6.

Figura 5.5 – Modal para cadastramento de produto no cardápio.

Fonte: Captura de tela realizada pelo autor.

É importante destacar que a direita do nome do produto adicionado ao cardápio, um botão com o ícone de uma lixeira é exibido. Clicar neste botão exclui o produto do cardápio do

Figura 5.6 – Produto adicionado ao cardápio.

Fonte: Captura de tela realizada pelo autor.

tenant. Após o reconhecimento da aplicação, foi feita uma análise no código fonte da página em busca de informações que poderiam vir a ser úteis, como listagem de diretórios, comentários e identificação de uso de bibliotecas em versões com vulnerabilidades conhecidas. A análise do código da página da aplicação sob a perspectiva do navegador é exibida na Figura 5.7.

Nas últimas linhas de código da página princpal foi identificado o uso da biblioteca *jQuery*, que é uma biblioteca de funções *JavaScript* que interage com a página *HTML* com o objetivo de simplificar os *scripts* interpretados no navegador do cliente. Pelo código fonte da página ainda é possível detectar que o arquivo da biblioteca em questão está armazenado no endereço */resources/js/jquery.min.js*.

Na sequência o arquivo da biblioteca *jQuery* foi acessado para análise. Nas primeiras linhas é possível identificar sua versão, como destacado na Figura 5.8.

Posteriormente foram realizadas buscas por subdomínios e subsistemas como sistemas de *e-mail*. Primeiramente foi feita uma busca "manual" por meio de um comando que, para cada palavra armazenada em uma determinada lista com palavras frequentemente utilizadas em subdomínios, faz uma tentativa de conexão com um endereço que é resultado da concatenação da palavra em questão com o restante do domínio da página da aplicação. Com o código de resposta *HTTP* recebido desta tentativa de conexão é possível determinar se aquele subdomínio existe. O código utilizado é exibido abaixo:

Figura 5.7 – Listagem do código da página inicial da aplicação *iCardapio*.

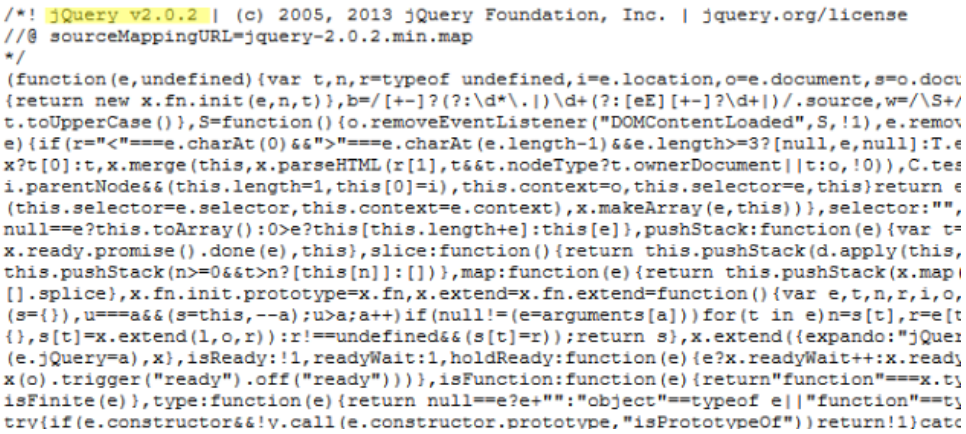
Fonte: Captura de tela realizada pelo autor.

```
for palavra in $(cat lista);do host $palavra.enderecodaaplicacao.com;done
    ↪  |grep has address | sort -u
```

Nenhum subdomínio foi encontrado com a execução do comando acima. Para finalizar a etapa de coleta de informações, foi realizada uma varredura por arquivos e diretórios. Para isso foi utilizada a ferramenta *Dirb* com o seguinte comando:

```
dirb http://endereco.com /usr/share/dirb/wordlists/big.txt
```

Figura 5.8 – Trecho do arquivo da biblioteca *jQuery*.



Fonte: Captura de tela realizada pelo autor.

Vários resultados foram encontrados mas, acessando os endereços indicados como arquivos ou diretórios, a página acessada era sempre a inicial ou uma página que acredita-se ser a página *template* da página do *tenant*, conforme mostrado na Figura 5.9. Esta suposição baseia-se no fato de que as páginas detectadas pela ferramenta *Dirb* não possuem sequer os dados básicos requisitados na criação de um novo cardápio.

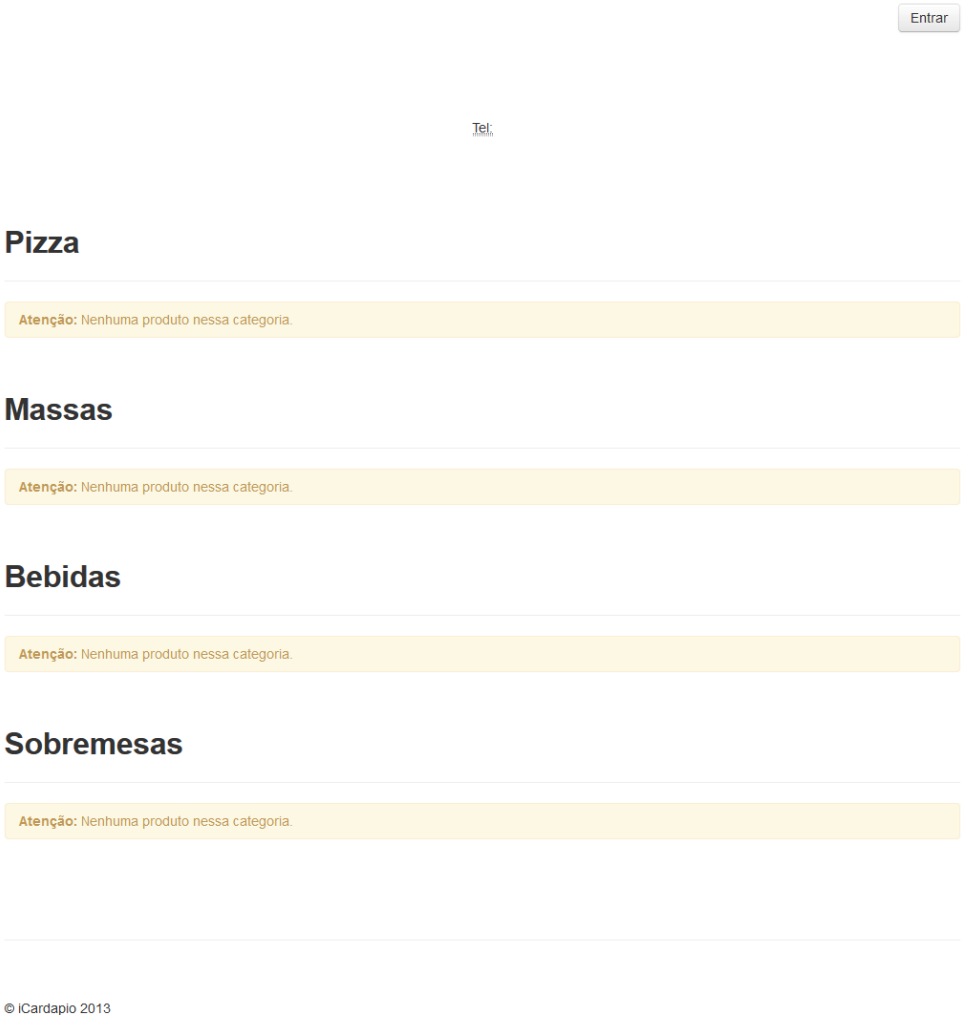
Estágio 3. Modelagem de Ameaças

Com base no que foi analisado na etapa anterior, acredita-se que a aplicação cria uma página para cada *tenant*, a qual é possível acessar pelo endereço da aplicação seguido do nome do *tenant*. Nesta página são exibidos os dados fornecidos pelo *tenant* no cadastro inicial, e todos os produtos cadastrados no seu cardápio.

Em um *Pentest* é importante identificar pontos de entrada para exploração de vulnerabilidades. Na aplicação testada, os únicos campos de inserção de dados identificados foram os campos para cadastro inicial, campos para entrar no painel administrativo, e campos para inserir novos produtos. Além dos campos de inserção, foi identificado um botão para remoção de produto do cardápio.

Com base em todas essas informações foram elaboradas duas estratégias de ataque: *Cross-site Scripting*, e *SQL Injection*. A primeira, *Cross-site Scripting*, pode ser aplicada na página inicial, afetando todos os clientes que fossem criar algum cardápio (criar novo *tenant*). A segunda estratégia seria conseguir acesso aos dados dos *tenants* por meio de exploração de uma possível vulnerabilidade de *SQL Injection*. Desta forma seria possível modificar, criar e excluir

Figura 5.9 – Página *template* de cardápio.



Fonte: Captura de tela realizada pelo autor.

dados de um ou até mesmo de todos os *tenants* por uma única porta de acesso, dependendo da abordagem adotada para administrar os dados.

Estágio 4. Análise de vulnerabilidades

Na etapa de coleta de informações, uma das informações coletadas foi que a aplicação utilizava a biblioteca *jQuery* na versão 2.0.2. Com a versão da biblioteca detectada, foi realizada uma busca em bases de dados de informações sobre segurança e vulnerabilidades conhecidas, como o *CVE Details*[14]. Desta maneira, foram encontrados indícios de que a versão da biblioteca *jQuery* utilizada na aplicação seria vulnerável à ataques do tipo *XSS*, como exposto na Figura 5.10.

[14] https://www.cvedetails.com/

Figura 5.10 – Vulnerabilidade encontrada na biblioteca *jQuery*.

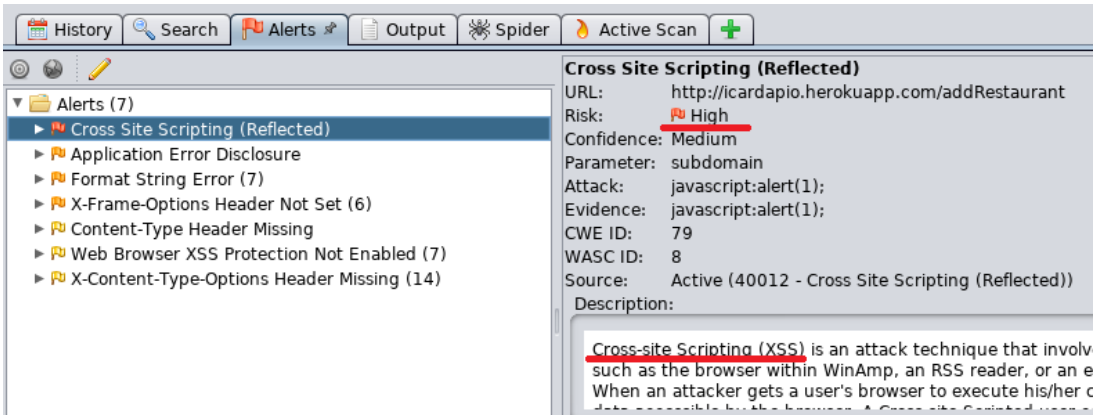
#	CVE ID	CWE ID	# of Exploits	Vulnerability Type(s)	Publish Date	Update Date	Score	Gained Access Level	Access	Complexity	Authentication	Conf.	Integ.	Avail.
1	CVE-2019-11358	79		XSS	2019-04-19	2019-05-10	4.3	None	Remote	Medium	Not required	None	Partial	None

jQuery before 3.4.0, as used in Drupal, Backdrop CMS, and other products, mishandles jQuery.extend(true, {}, ...) because of Object.prototype pollution. If an unsanitized source object contained an enumerable __proto__ property, it could extend the native Object.prototype.

Fonte: Captura de tela realizada pelo autor.

Em busca de mais vulnerabilidades foi feita uma varredura com a ferramenta *OWASP ZAP*. Alguns alertas foram emitidos e, dentre eles, um alerta de alto risco. O alerta em questão é um alerta sobre *Cross-site Scripting*. A ferramenta ainda forneceu uma descrição da vulnerabilidade e um exemplo de como ela pode ser explorada. A detecção da vulnerabilidade pela ferramenta é exibida na Figura 5.11.

Figura 5.11 – Vulnerabilidade encontrada pela ferramenta *OWASP ZAP*.



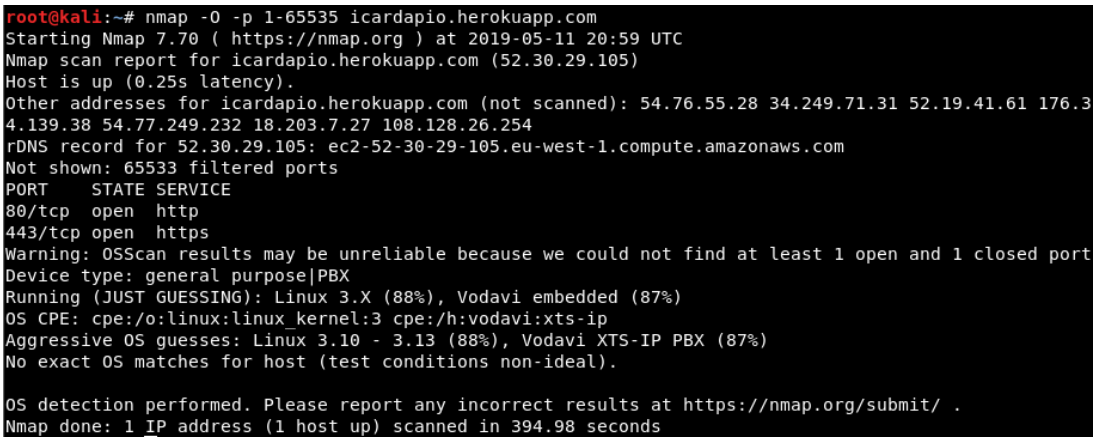
Fonte: Captura de tela realizada pelo autor.

Em seguida foi feita uma varredura de portas com a ferramenta *Nmap* com o objetivo de descobrir portas abertas e serviços sendo executados na aplicação. Os resultados obtidos pela varredura é apresentado na Figura 5.12.

Esta varredura foi feita com o propósito de identificar serviços por meio de portas abertas. Entretanto, o fato de mais portas não estarem abertas não significa que não há mais serviços sendo executados na aplicação. Na Figura 5.13 é possível ver que a porta 3306, porta padrão do serviço *mysql*, encontra-se no estado *filtered*, o que significa que o *Nmap* não consegue determinar se a porta está aberta porque uma filtragem de pacotes impede que as sondagens alcancem a porta. Um serviço de *Firewall* pode ser responsável pela filtragem de pacotes.

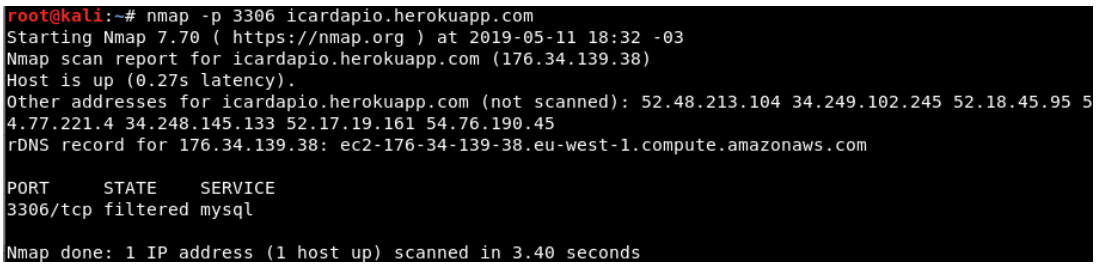
Apesar de as varreduras não terem encontrado serviços *SQL* executando no servidor da aplicação, foram aplicadas técnicas manuais para verificar se a aplicação estaria vulnerável a *SQL Injection*, como preencher campos de inserção com códigos *SQL* com o intuito de

Figura 5.12 – Varredura executada pela ferramenta *Nmap*.



Fonte: Captura de tela realizada pelo autor.

Figura 5.13 – Varredura focada na porta 3306.



Fonte: Captura de tela realizada pelo autor.

confundir o interpretador, conforme exemplificado no Capítulo 2. Não foram encontradas vulnerabilidades a *SQL Injection*.

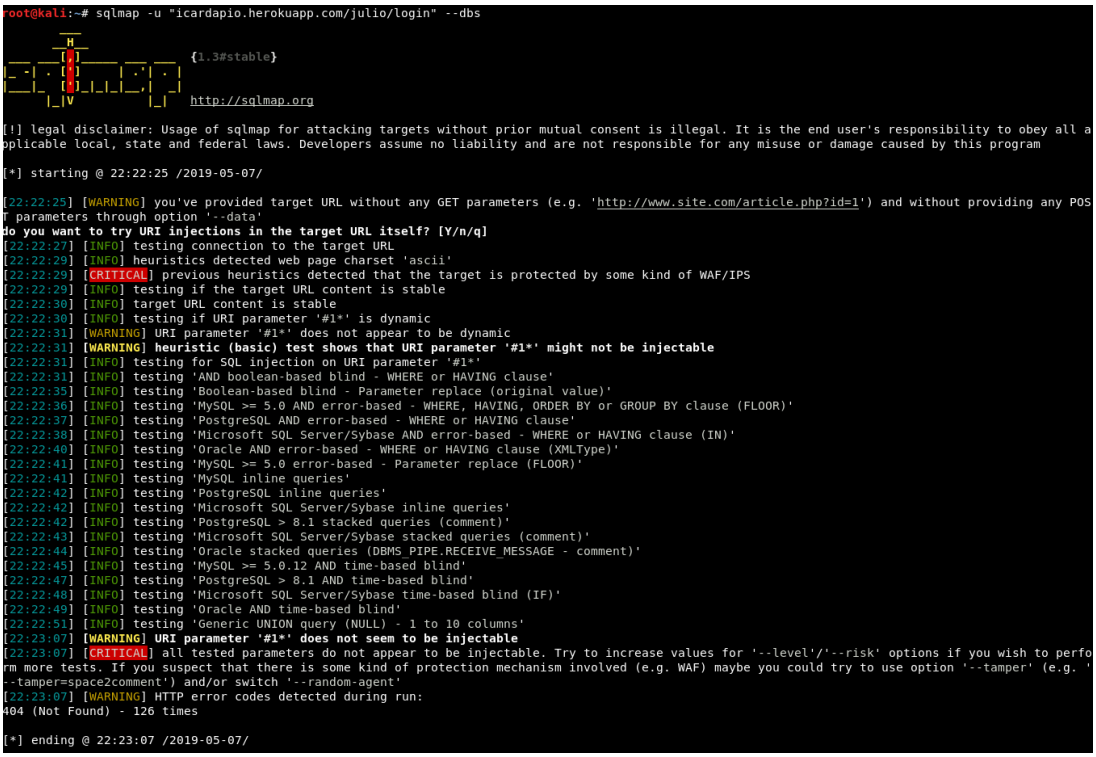
Além da análise manual, também foi executada a ferramenta *Sqlmap* para buscar por vulnerabilidades de *SQL Injection*. Na Figura 5.14 são exibidos os resultados desta varredura. Nenhuma vulnerabilidade de *SQL Injection* foi encontrada com a ferramenta.

Estágio 5. Exploração

A única vulnerabilidade encontrada na aplicação foi a de *Cross-site Scripting* (*XSS*). Como exposto anteriormente, esta vulnerabilidade permite que códigos sejam inseridos e executados na página devido à uma falha, ou falta de verificação dos dados inseridos na aplicação. O primeiro código inserido na aplicação foi um código para testar efetivamente a vulnerabilidade a *XSS*. O código inserido é exibido a seguir:

```
<script>alert(document.cookie)</script>
```

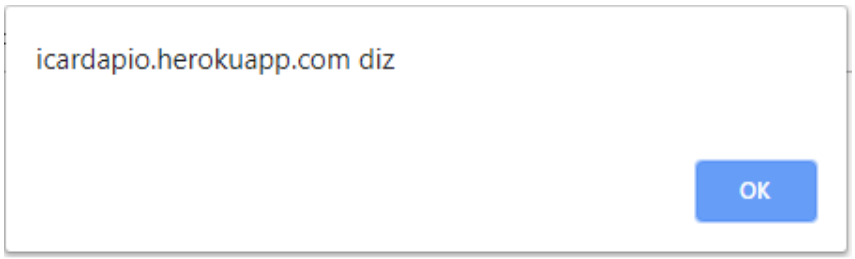
Figura 5.14 – Varredura executada pelo *Sqlmap*.



Fonte: Captura de tela realizada pelo autor.

O código acima faz com que seja exibido, por meio de um *alert*, o *cookie*. Este código foi inserido com o propósito de capturar o *cookie* da sessão para tentar realizar um ataque de *Cross-site Request Forgery*. O resultado é exibido na Figura 5.15.

Figura 5.15 – Resultado da execução do *script* que exibe na tela o *cookie*.



Fonte: Captura de tela realizada pelo autor.

O resultado do *script* foi um *alert* em branco. Isto provavelmente se deve ao fato de a aplicação utilizar *cookies* do tipo *HttpOnly*, que são *cookies* inacessíveis pela *API Javascript Document.cookie*. Este tipo de *cookie* é enviado apenas para o servidor.

Outro tipo de ataque que poderia ser explorado devido a vulnerabilidade *Cross-site Scripting* é o *Cross-site Request Forgery (CSRF)*. No entanto, para que seja possível utilizar um ataque deste tipo, é necessário que exista na aplicação alguma URL com efeito colateral,

como troca de senha, transferência bancária, troca de *e-mail*, etc. Como foi exibido na etapa coleta de informações do planejamento, não existe uma URL com efeito colateral para que este tipo de ataque possa ocorrer com o propósito de ter acesso aos dados dos múltiplos *tenants* da aplicação. As únicas submissões de formulários onde este tipo de ataque poderia ser testado na aplicação *iCardapio* são nas submissões de informações sobre produtos a serem adicionados ao cardápio, ou na criação de novo cardápio/*tenant*.

Outra abordagem levada em consideração foi injetar código *SQL* por meio de códigos *Javascript* também injetados na página pela vulnerabilidade *Cross-site Scripting*, mas se as consultas *SQL* da aplicação não são vulneráveis a *SQL Injection*, então códigos *Javascript* não farão com que elas se tornem vulneráveis. Portanto esta abordagem também foi descartada.

Alguns outros *scripts* foram inseridos por meio da vulnerabilidade de *Cross-site Scripting*, como o *script* apresentado a seguir:

```
<script>
var elements = document.getElementsByClassName(container);
while(elements.length > 0){
elements[0].parentNode.removeChild(elements[0]);
}
</script>
```

O ataque executado com o *script* citado acima foi realizado com sucesso, excluindo todo o conteúdo contido na *div "container"*, apagando todo o conteúdo da página do *tenant*. Este ataque é perigoso e pode acarretar em graves consequências dependendo do ambiente, mas, neste caso, não atinge o objetivo de conceder acesso aos dados dos múltiplos *tenants* ao atacante.

Estágio 6. Pós-exploração

Nesta etapa atentou-se para as tentativas de obtenção de mais informações sobre o sistema atacado com a análise de arquivos, análise de dados armazenados nos bancos de dados, aumento de privilégios, etc. Porém, nenhum ataque realizado na etapa anterior atingiu o objetivo de conseguir acesso aos dados dos *tenants*, pois não havia vulnerabilidade que atingisse esta camada da aplicação. Portanto, não foi executada a etapa Pós-exploração neste caso de teste de invasão.

Estágio 7. Relatórios

No Experimento 1 não pôde ser realizado com sucesso devido à falta de vulnerabilidade que permitisse acesso aos dados. Portanto, não havendo uma falha que permita acesso aos dados da aplicação, não foi possível medir o nível de exposição dos dados. Entretanto, buscou-se explorar as vulnerabilidades desta aplicação por meio de ferramentas e tentativas manuais.

Vários métodos para garantir a proteção dos dados foram adotados na aplicação testada no Experimento 1, como utilização da *flag httpOnly* nos *cookies*, o que previne ataques do tipo *CSRF*. No entanto, a vulnerabilidade *XSS* foi encontrada e, apesar de não permitir acesso indevido aos dados dos *tenants*, permite a execução de ataques que afetam o funcionamento da página do *tenant*.

Ressalta-se que só foi possível explorar a vulnerabilidade *XSS* com sessão ativa do administrador da página do *tenant*, pois não há campos em que usuários possam inserir dados. Na ocorrência de qualquer alteração na aplicação que passa a permitir inserção de dados por parte dos usuários, a vulnerabilidade *XSS* deverá ser tratada imediatamente pois, como demonstrado nos experimentos, permitiria execução de ataques danosos.

5.3 Experimento 2 - *MtShop*

Estágio 1. Pré-engajamento

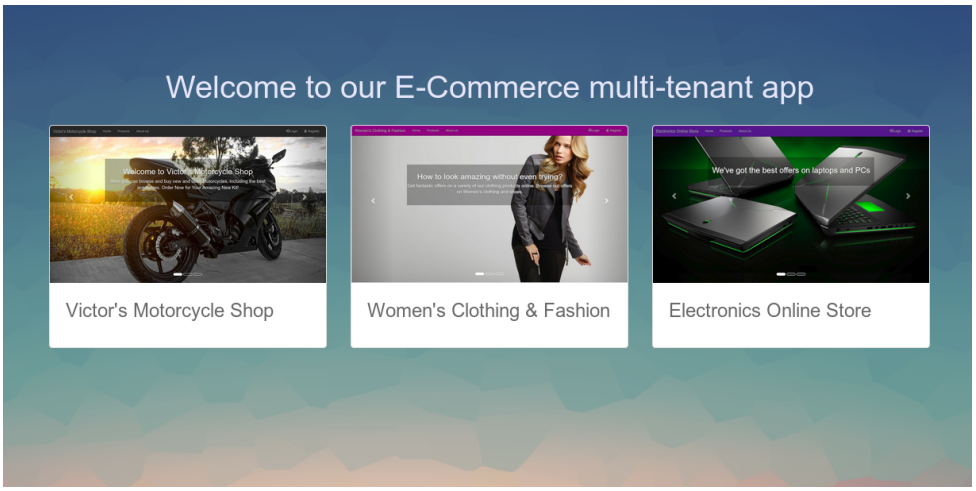
As orientações dadas ao aplicador dos testes foram acerca do endereço web da aplicação, o fato de que esta instância do sistema existia apenas para este fim e o usuário e senha padrão de um dos três *tenants*.

Estágio 2. Coleta de informações

Com o endereço web da aplicação em mãos, os primeiros passos dados foram o de reconhecimento da aplicação. Primeiramente a aplicação alvo foi acessada e analisada. Foi observado que a página inicial da aplicação exibia três lojas (*tenants*) criadas na aplicação, como exibido na Figura 5.16.

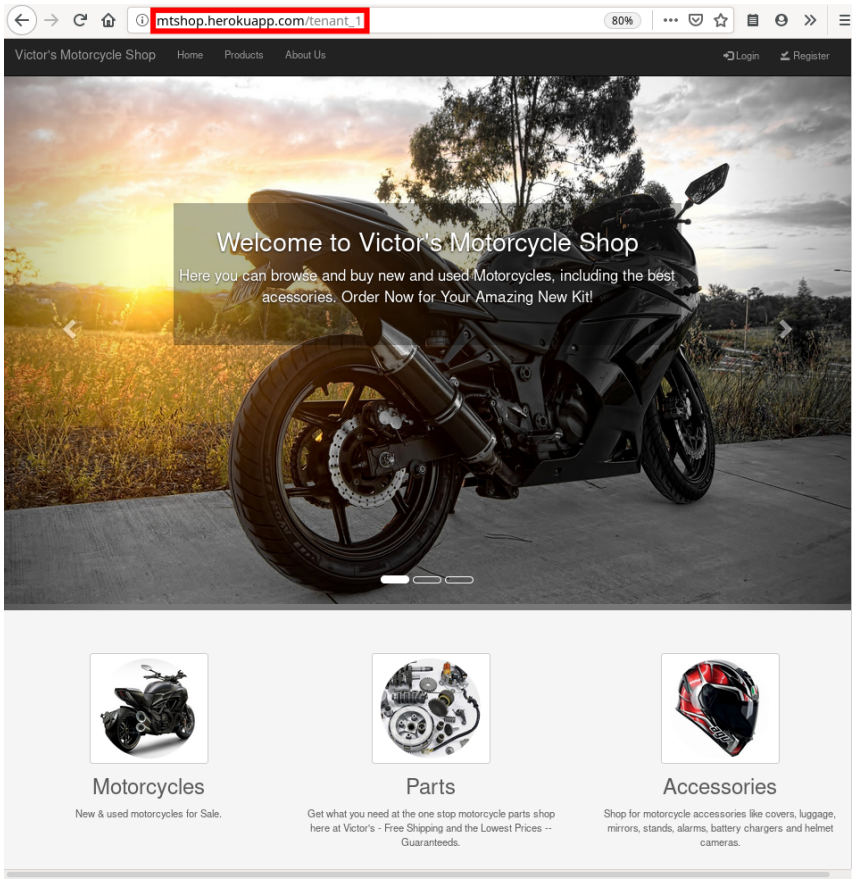
Clicando nos links das lojas, somos redirecionados para suas respectivas páginas iniciais. Nota-se, como mostrado na Figura 5.17, que o *MtShop* adota a abordagem de identificação dos *tenants* que adiciona o *tenant ID* na URI.

Figura 5.16 – Página inicial da aplicação *MtShop*.



Fonte: Captura de tela realizada pelo autor.

Figura 5.17 – Página inicial do *Tenant* 1.

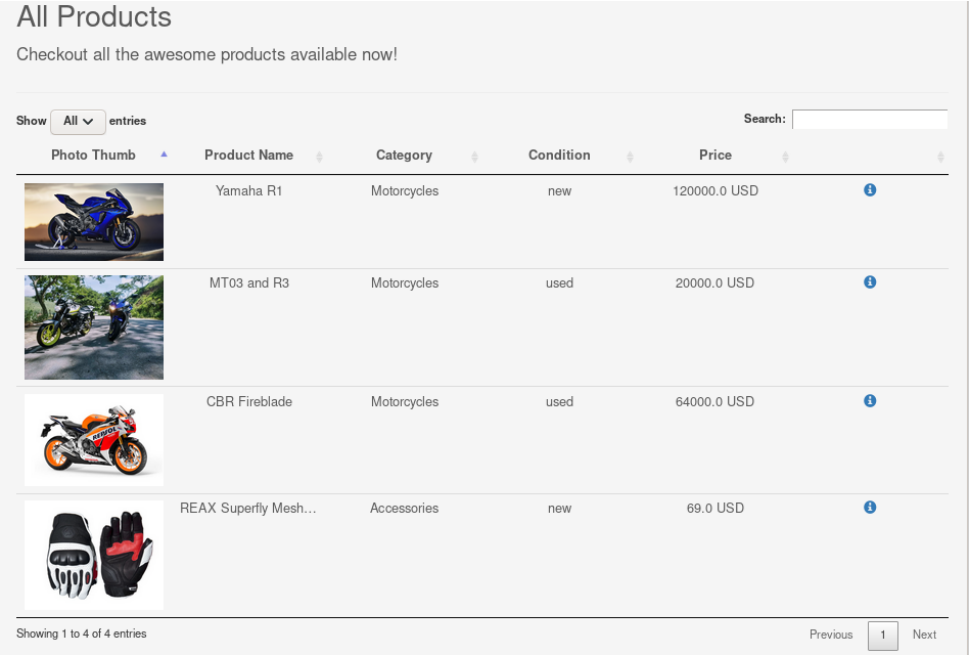


Fonte: Captura de tela realizada pelo autor.

Na parte superior da página inicial de cada *tenant* é exibido um menu com o nome da loja e alguns botões. Abaixo é listado todos os botões desse menu com suas respectivas funções e capturas de tela:

- *Home*: Link para a página inicial do *tenant*.
- *Products*: Link para página que exibe todos os produtos cadastrados na loja. A página contém, além das informações sobre os produtos cadastrados, um campo para busca de produtos, campo para selecionar a quantidade de produtos exibidos na página, botões para navegar pelas páginas de produtos cadastrados e, à direita das informações de cada produto, um botão para ir para a página do produto em questão. Figura 5.18. Ao clicar no botão o usuário é redirecionado para uma página com o seguinte endereço: *mtshop.herokuapp.com/**nome do tenant**/product/viewProduct/**número do produto***.

Figura 5.18 – Página *Products*



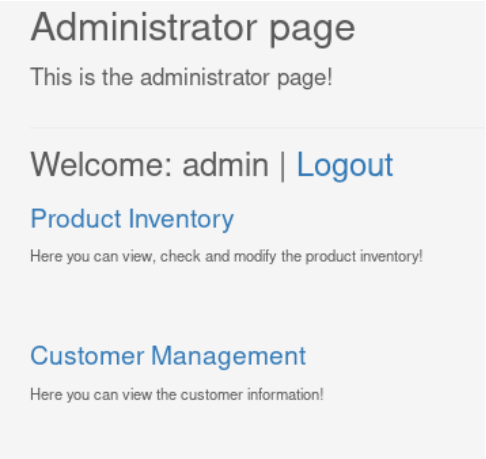
Fonte: Captura de tela realizada pelo autor.

- *About Us*: Página com informações sobre a loja.
- *Login*: Página com formulário para login.
- *Register*: Página com formulário para registrar novo usuário de cliente.

Além dos botões contidos no menu da parte superior da página inicial dos *tenants*, existem três links no seguimento inferior da mesma pelos quais é possível acessar os produtos cadastrados por categorias.

No Apêndice A são apresentadas outras telas e funcionalidades da aplicação.

Figura 5.19 – Página exibida ao clicar no botão *Admin*.



Fonte: Captura de tela realizada pelo autor.

Após efetuar o login na aplicação, o botão *Admin* é exibido na barra de menu da página. Ao clicar nesta opção, o usuário é direcionado para a página exibida na Figura 5.19.

Ao clicar na opção *Customer Management* exibe uma página com informações sobre todos os usuários cadastrados na aplicação.

Após o reconhecimento da aplicação, foi feita uma análise no código fonte da página em busca de informações que poderiam vir a ser úteis, como listagem de diretórios, comentários e identificação de uso de bibliotecas em versões com vulnerabilidades conhecidas. Foi identificado a utilização de algumas bibliotecas, como exposto na Figura 5.20. Entretanto, analisando as versões de cada uma dessas bibliotecas, não foram encontradas vulnerabilidades conhecidas além das já encontradas nas fases anteriores.

Posteriormente foram realizadas buscas por subdomínios e subsistemas como sistemas de *e-mail* da mesma forma que no experimento anterior. Nenhum subdomínio foi encontrado. Para finalizar a etapa de coleta de informações, foi realizada uma varredura por arquivos e diretórios. Para isso foi realizada uma varredura com a ferramenta *Dirb* da mesma forma que no experimento anterior.

A varredura executada com a ferramenta *Dirb* encontrou 17 objetos. Entretanto, como apresentado na Figura 5.21, a resposta *HTTP* à tentativa de acesso a estes arquivos foi o código 500, indicando que foi encontrada uma condição inesperada que o impediu de atender a solicitação. Não é possível acessar estes objetos e eles são, possivelmente, arquivos de configuração do ambiente em que a aplicação está hospedada.

Figura 5.20 – Análise do código da página inicial do *tenant 1* da aplicação MtShop.

Fonte: Captura de tela realizada pelo autor.

Estágio 3. Modelagem de Ameaças

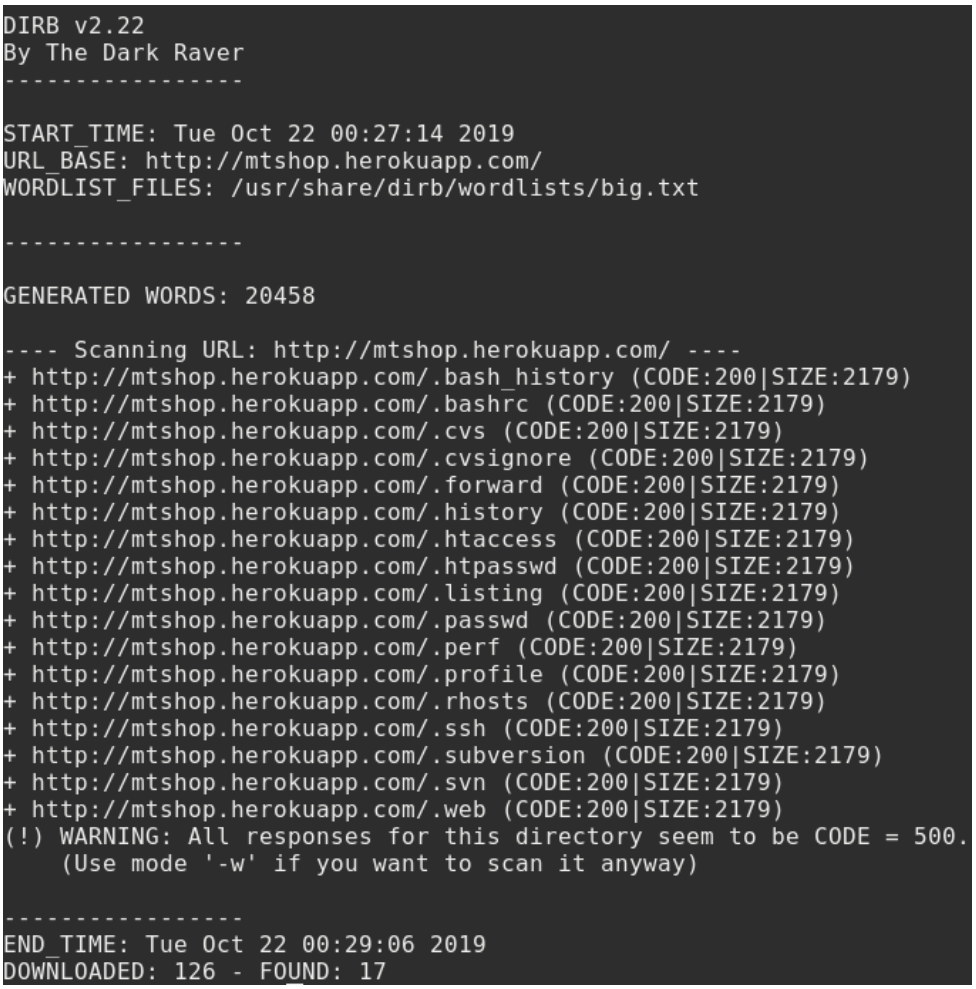
Com base no que foi analisado na etapa anterior, acredita-se que existam três páginas, uma para cada *tenant*. A instância de cada *tenant* possui um identificador único e é acessada pela URI que contem o respectivo *tenant ID*. A página de cada *tenant* é uma loja virtual que segue uma estrutura padrão, porém é possível perceber que cada uma das lojas vende um tipo específico de produto.

Como mencionado no Experimento 1, em um *Pentest* é importante identificar pontos de entrada para exploração de vulnerabilidades, e muitas vezes esses pontos de entrada podem ser campos nos quais são possíveis inserir dados. Nos testes do Experimento 2 foram identificados alguns campos para inserção de dados por parte dos usuários:

- Campos na tela de cadastramento de cliente.
- Campos para entrar no sistema (cliente ou administrador (*tenant*).
- Campos para inserir ou modificar produtos cadastrados no sistema.
- Campo de pesquisa de produtos.

Assim como no Experimento 1, as informações coletadas nas etapas anteriores resultaram na elaboração de duas estratégias de ataque: *Cross-site Scripting*, e *SQL Injection*. A primeira, *Cross-site Scripting*, pode ser aplicada nas informações sobre produtos ou novo usuário,

Figura 5.21 – Resultado da varredura executada com o *Dirb*.



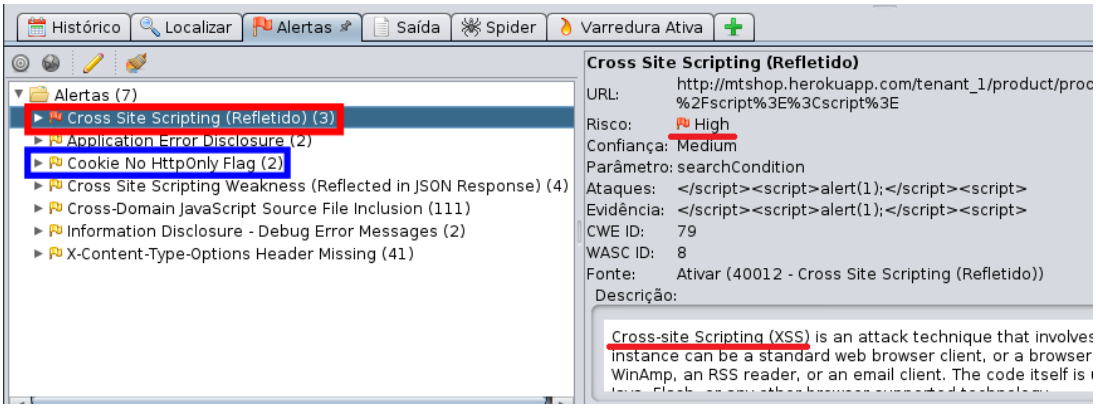
Fonte: Captura de tela realizada pelo autor.

afetando todos os clientes que fossem acessar a página correspondente ao produto em questão, ou o administrador que acessar a página com as informações sobre os clientes. A segunda estratégia seria conseguir acesso aos dados dos *tenants* por meio de exploração de uma possível vulnerabilidade de *SQL Injection*. Desta forma seria possível iniciar sessão sem o conhecimento do *login* ou senha, ou até mesmo modificar, criar e excluir dados dos *tenants*.

Estágio 4. Análise de vulnerabilidades

Em busca de mais vulnerabilidades uma varredura foi executada com a ferramenta *OWASP ZAP*. Alguns alertas foram emitidos e, dentre eles, um alerta de alto risco. O alerta em questão é um alerta sobre *Cross-site Scripting*. A ferramenta ainda forneceu uma descrição da vulnerabilidade e um exemplo de como ela pode ser explorada. A detecção da vulnerabilidade pela ferramenta é exibida na Figura 5.22.

Figura 5.22 – Vulnerabilidade encontrada pela ferramenta *OWASP ZAP*.



Fonte: Captura de tela realizada pelo autor.

Além do alerta relativo a vulnerabilidade *XSS*, foi emitido o alerta *Cookie No HttpOnly Flag*, como exibido na Figura 5.22. *Cookies HttpOnly* não são acessíveis pela *API Javascript Document.cookie;*, o que impossibilitou o avanço na exploração da vulnerabilidade *XSS* no Experimento 1.

Em seguida foi feita uma varredura de portas com a ferramenta *Nmap* com o objetivo de descobrir portas abertas e serviços sendo executados na aplicação. Os resultados obtidos foram os mesmos do experimento anterior, como apresentado na Figura 5.23.

Figura 5.23 – Varredura executada pela ferramenta *Nmap*.

```
Starting Nmap 7.80 ( https://nmap.org ) at 2019-10-17 11:30 -03
Nmap scan report for mtshop.herokuapp.com (35.170.37.11)
Host is up (0.14s latency).
Other addresses for mtshop.herokuapp.com (not scanned): 100.26.7.15 34.238.112.56 34.198.105.97 52.20
0.110.231 3.224.173.25 3.225.95.126 52.71.166.89
rDNS record for 35.170.37.11: ec2-35-170-37-11.compute-1.amazonaws.com
Not shown: 65533 filtered ports
PORT    STATE SERVICE
80/tcp  open  http
443/tcp open  https
Warning: OSScan results may be unreliable because we could not find at least 1 open and 1 closed port
Device type: general purpose|PBX
Running (JUST GUESSING): Linux 3.X (88%), Vodavi embedded (87%)
OS CPE: cpe:/o:linux:linux_kernel:3 cpe:/h:vodavi:xts-ip
Aggressive OS guesses: Linux 3.10 - 3.13 (88%), Vodavi XTS-IP PBX (87%)
No exact OS matches for host (test conditions non-ideal).

OS detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 196.55 seconds
```

Fonte: Captura de tela realizada pelo autor.

Apesar das varreduras não terem encontrado serviços *SQL* executando no servidor da aplicação, foram aplicadas técnicas manuais para verificar se a aplicação estaria vulnerável a *SQL Injection*, como preencher campos de inserção com códigos *SQL* com o intuito de confundir o interpretador, conforme exemplificado anteriormente. Não foram encontradas vulnerabilidades a *SQL Injection*.

Além da análise manual, também foi executada a ferramenta *Sqlmap* para buscar por vulnerabilidades de *SQL Injection*.

Estágio 5. Exploração

A única vulnerabilidade encontrada na aplicação foi a vulnerabilidade de *Cross-site Scripting* (*XSS*). O primeiro código inserido na aplicação foi um código para testar efetivamente a vulnerabilidade a *XSS*. O código inserido é exibido a seguir:

```
<script>alert(document.cookie)</script>
```

O código acima faz com que seja exibido, por meio de um *alert*, o *cookie*. Este código foi inserido com o propósito de capturar o *cookie* da sessão para tentar realizar um ataque de *Cross-site Request Forgery*. O resultado é exibido na Figura 5.24.

Figura 5.24 – Resultado da execução do *script* que exibe na tela o *cookie*.

Fonte: Captura de tela realizada pelo autor.

Após a constatação de que o *cookie* está disponível, iniciou-se a formulação de uma hipótese na qual seria possível explorar a vulnerabilidade. Os passos necessários, hipotéticos, são descritos a seguir:

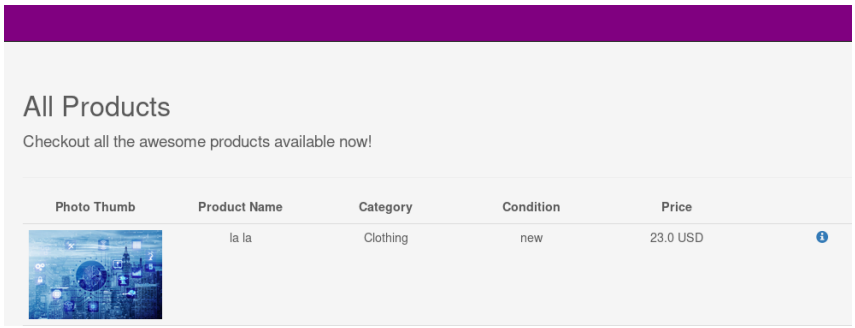
1. Invasor cria usuário na aplicação do *tenant* 1 e preenche uma das informações com um código malicioso que o envia o *cookie*.

2. *Tenant* 1 inicia sessão e acessa a página na qual acessa as informações dos clientes cadastrados. O *script* é executado.

3. Invasor utiliza as informações do *cookie* para enviar uma requisição para a aplicação em nome do *tenant* autenticado, mas efetuando uma operação na aplicação do outro *tenant*.

Para simular os passos descritos acima, planejou-se utilizar uma sessão ativa do *tenant* 1 para excluir o produto mostrado na Figura 5.25, que pertence ao *tenant* 2.

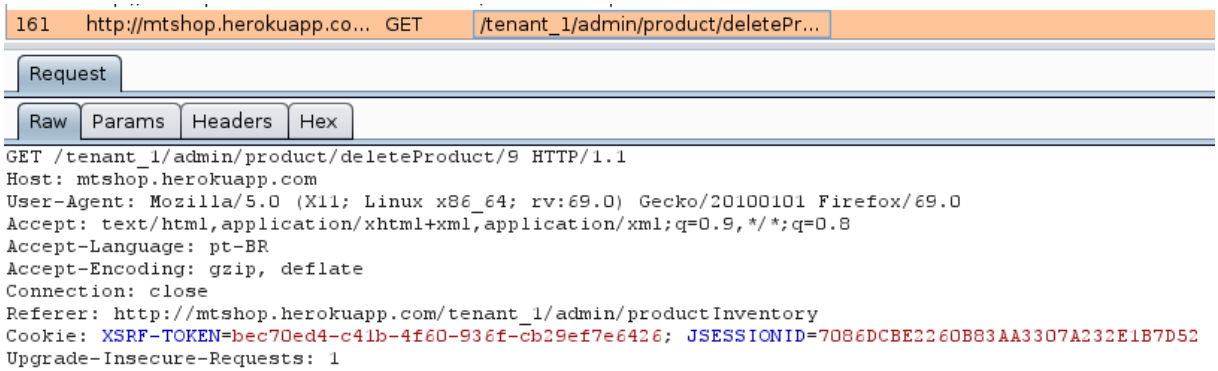
Figura 5.25 – Produto que deve ser excluído com a exploração da vulnerabilidade.



Fonte: Captura de tela realizada pelo autor.

Nos passos seguintes utilizou-se a aplicação *Burp Suite*. Com o administrador do *tenant* 1 em sessão ativa, iniciou-se a interceptação dos dados do navegador e foi enviado uma requisição para excluir um produto do *tenant* 1. O produto foi excluído e a requisição foi capturada pelo *Burp Suite*, como mostrado na Figura 5.26.

Figura 5.26 – Captura de requisição de exclusão de produto.

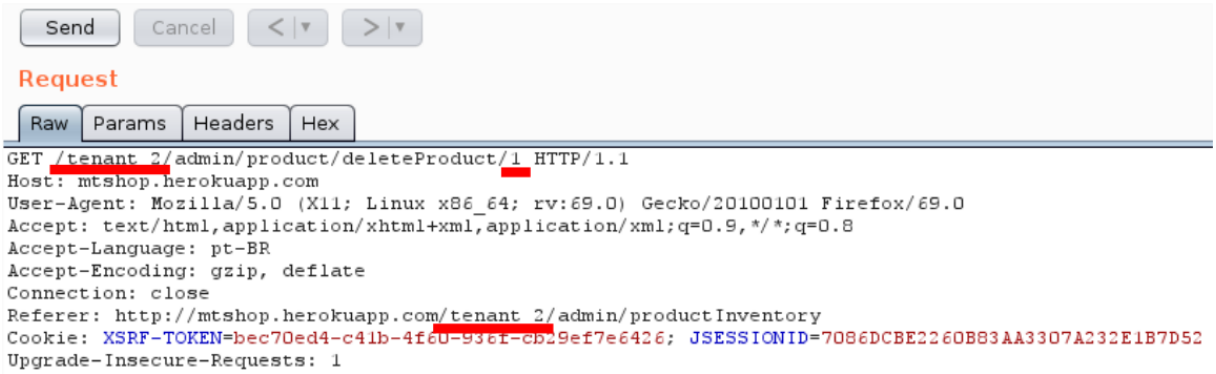


Fonte: Captura de tela realizada pelo autor.

Logo após a captura, o código foi enviado para o *Repeater* da aplicação e modificado em dois pontos: no endereço *GET* e no endereço *Referer* para que atinja o *tenant* 2. A requisição modificada é exibida na Figura 5.27.

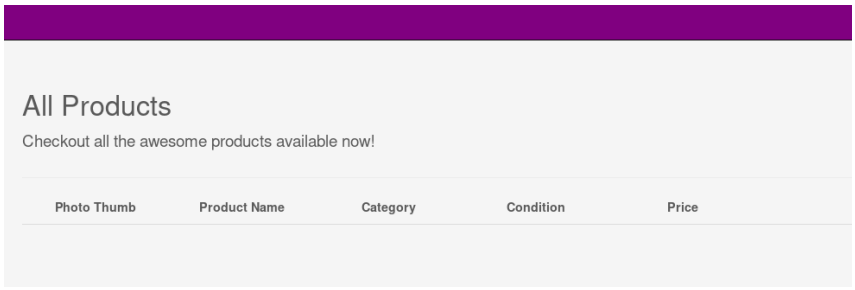
Ao clicar na opção *Send* do *Burp Suite*, a requisição é enviada ao servidor da aplicação **multi-tenant**. Após o envio da requisição, a página dos produtos do *tenant* 2 foi acessada e o produto havia sido excluído, conforme mostrado na Figura 5.28.

Figura 5.27 – Requisição modificada.



Fonte: Captura de tela realizada pelo autor.

Figura 5.28 – Página de produtos do *tenant* 2.



Fonte: Captura de tela realizada pelo autor.

Estágio 6. Pós-exploração

Nenhum ataque realizado na etapa anterior atingiu o objetivo de conseguir acesso aos dados dos *tenants*, pois não havia vulnerabilidade que atingisse esta camada da aplicação. Portanto, não foi executada a etapa Pós-exploração neste caso de teste de invasão.

Estágio 7. Relatórios

Não foram encontradas vulnerabilidade que permitisse acesso aos dados. Apesar de não ser possível obter acesso aos dados dos *tenants*, foi possível executar ataques que causam dano à qualquer *tenant* por meio de um ataque realizado a um único *tenant*. No contexto de prevenir acesso indevido aos dados, medidas de segurança como adoção de boas práticas de programação que evitam vulnerabilidades como *SQL Injection* devem ser adotadas, fornecendo um grau razoável de proteção dos dados. Contudo, as vulnerabilidades de *Cross-Site Scripting* e *Cross-Site Request Forgery* foram encontradas na aplicação e devem ser tratadas para evitar danos como os expostos neste trabalho.

Para prevenir ataques do tipo *CSRF*, duas abordagens simples podem ser adotadas: a) incluir um *token* aleatório em cada solicitação, para que o sistema verifique se o formulário é válido comparando o *token* com o armazenado na variável sessão de usuário; b) usar nomes aleatórios para cada campo de formulário. Dessa forma o nome aleatório de cada campo é armazenado em uma variável de sessão, fazendo com que cada submissão do formulário gere um novo nome aleatório para o campo.

5.4 Experimento 3 - *ToyExample*

Estágio 1. Pré-engajamento

Por se tratar de experimentos em instâncias de aplicações dedicadas a esse fim, tais discussões e acordos foram realizados apenas entre os envolvidos no trabalho, como nos experimentos anteriores.

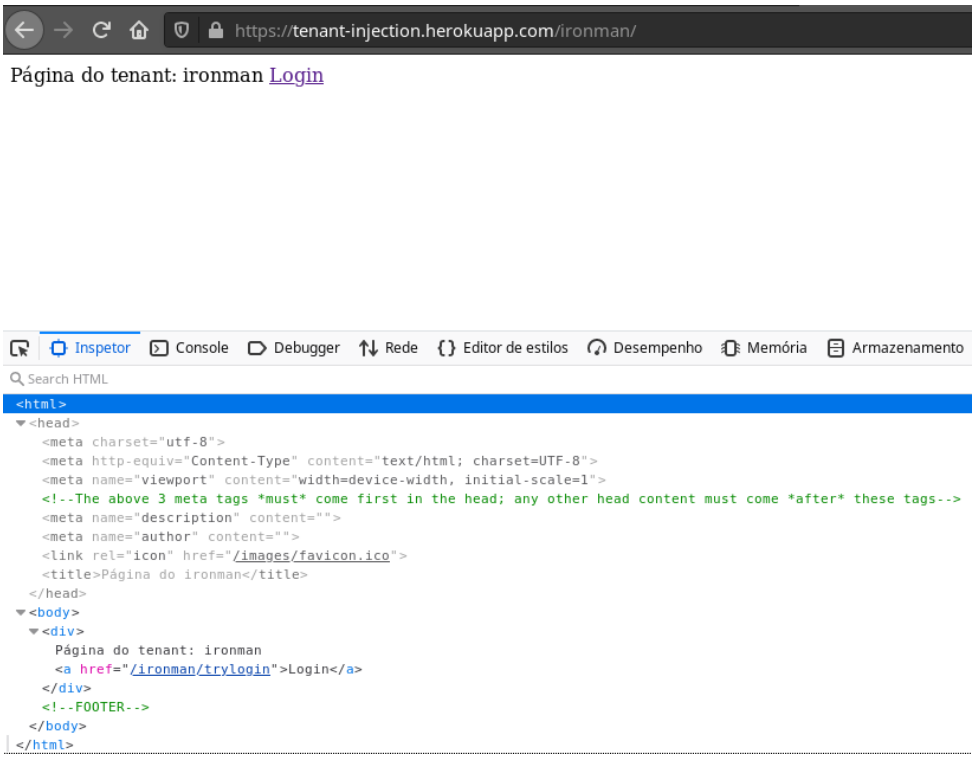
Estágio 2. Coleta de informações

A aplicação foi desenvolvida com cinco versões de *tenants*. Entretanto, realizou-se o *pentest* utilizando a URI de apenas uma instância *https://tenant-injection.herokuapp.com/ironman/*. Os primeiros passos dados foram o de reconhecimento da aplicação. Primeiramente a aplicação alvo foi acessada e analisada sem a presença do identificador do *tenant*. A página acessada não exibia nenhum conteúdo e a inspeção de seu código realizada pelo navegador confirmou que não havia nenhum dado a não ser a estrutura básica *HTML*. Em seguida, a URI com o identificador */ironman/* foi acessada e foi encontrada apenas uma mensagem informando o nome do *tenant* daquela instância e um link para a página de login, como representado na Figura 5.29.

Ao clicar no link *Login*, somos redirecionados para uma página com campos para inserção de usuário e senha, além do botão para realizar o login, como ilustrado na Figura 5.30. Posteriormente foram realizadas buscas por subdomínios e subsistemas como sistemas de e-mail da mesma forma que nos experimentos anteriores. Nenhum subdomínio foi encontrado. Para finalizar a etapa de coleta de informações, foi realizada uma varredura por arquivos e diretórios. Para isso foi realizada uma varredura com a ferramenta Dirb da mesma forma que nos experimentos anteriores.

Como exibido na Figura 5.31, a varredura executada com a ferramenta *Dirb* encontrou os mesmos 17 objetos encontrados no Experimento 2, além de outros dois itens. Os 17 primei-

Figura 5.29 – Página da aplicação *ToyExample*.



Fonte: Captura de tela realizada pelo autor.

ros objetos são, possivelmente, arquivos de configuração do ambiente em que a aplicação está hospedada. O objeto *error* foi acessado e nos leva a uma página em branco.

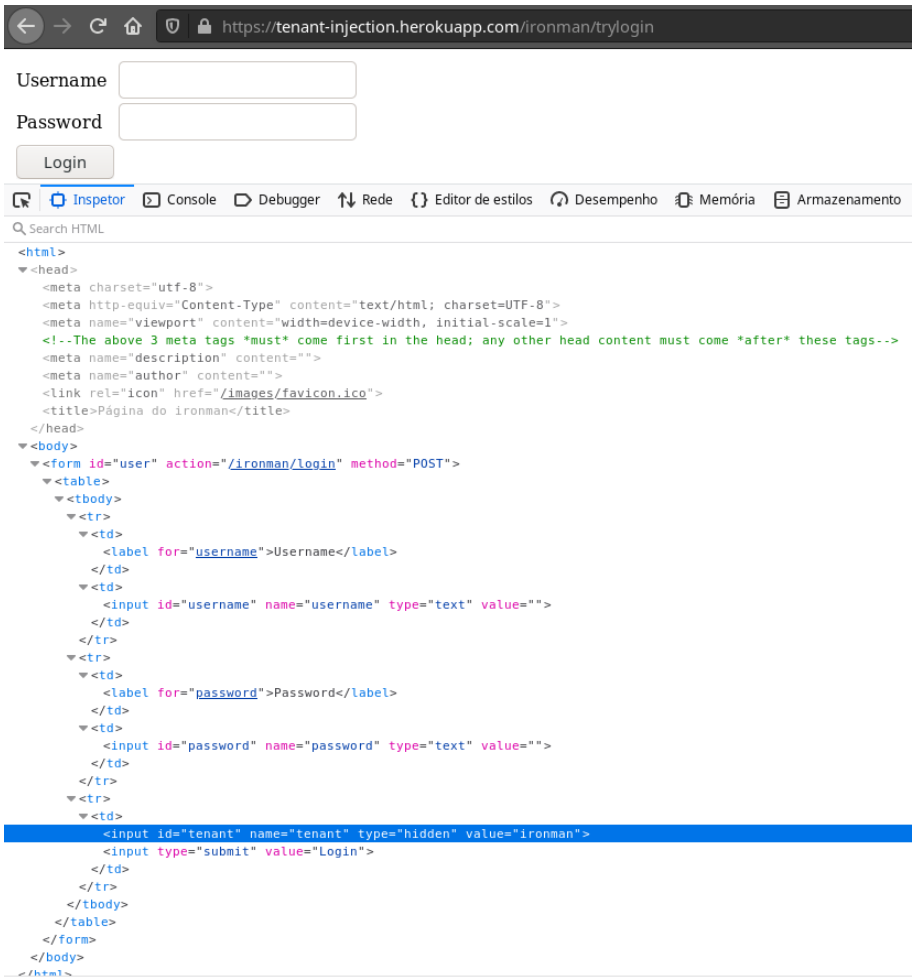
Estágio 3. Modelagem de Ameaças

Nos experimentos anteriores, o objetivo dos *pentests* era coletar informações das aplicações, modelar ameaças e, posteriormente, dar continuidade com a análise de vulnerabilidades e exploração. Neste experimento, entretanto, o objetivo foi tentar explorar vulnerabilidades de *SQL Injection* para avaliar o nível de exposição dos dados da aplicação, pois não foi possível explorar esta vulnerabilidade nos experimentos anteriores. Portanto, a estratégia de ataque definida para este experimento é explorar falhas de *SQL Injection* em busca de acesso ao banco de dados da aplicação.

Estágio 4. Análise de vulnerabilidades

Realizou-se uma varredura com a ferramenta *OWASP ZAP* e alguns alertas foram emitidos, dentre eles, um *SQL Injection* de alto risco. A ferramenta ainda forneceu uma descrição da

Figura 5.30 – Página de *login* da aplicação *ToyExample*.



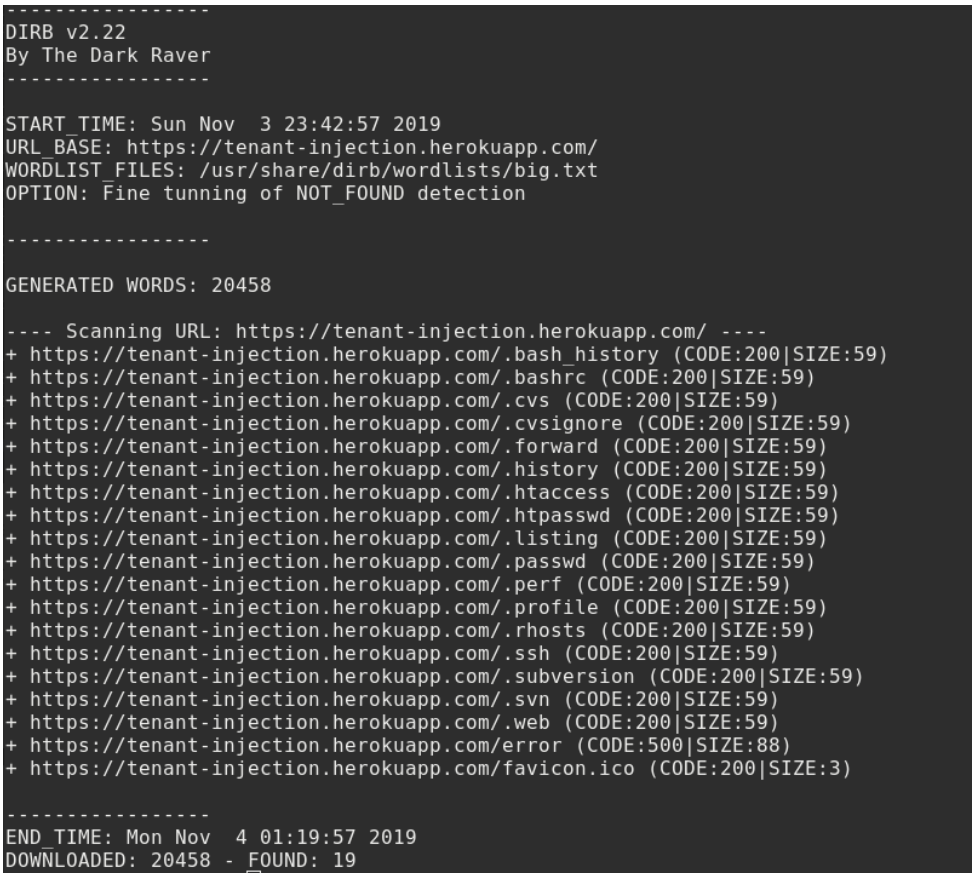
Fonte: Captura de tela realizada pelo autor.

vulnerabilidade e um exemplo de como ela pode ser explorada. A detecção da vulnerabilidade pela ferramenta *OWASP ZAP* é exibida na Figura 5.32.

Em seguida realizou-se uma varredura de portas com a ferramenta *Nmap* com o objetivo de descobrir portas abertas e serviços sendo executados na aplicação. Os resultados obtidos foram os mesmos dos dois experimentos anteriores e são apresentados na Figura 5.33

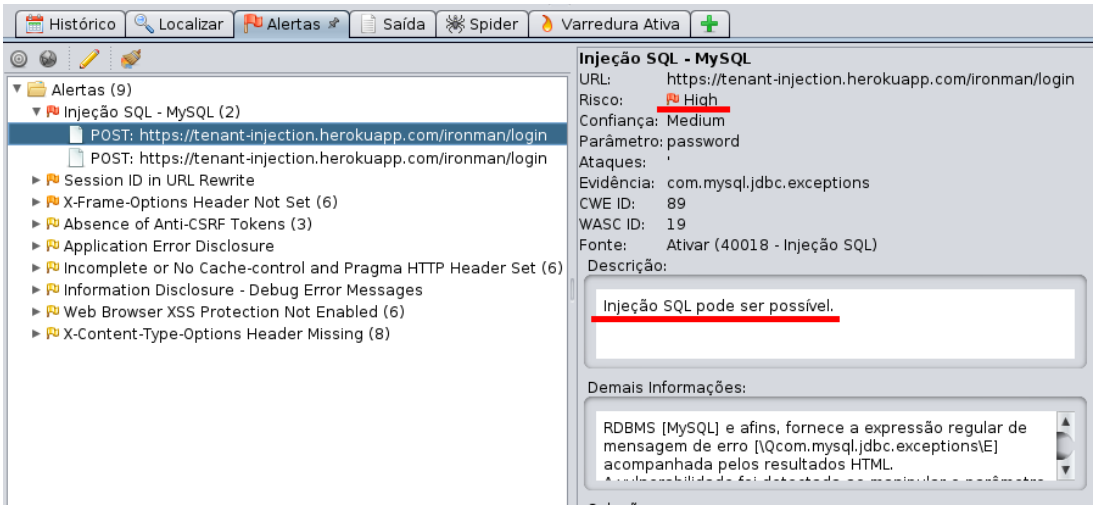
A ferramenta *Sqlmap* foi executada para buscar por vulnerabilidades de *SQL Injection*. Apesar de haver a vulnerabilidade à *SQL Injection* na aplicação, como foi mostrado com o uso da ferramenta *OWASP ZAP*, a ferramenta *Sqlmap* não conseguiu explorar a vulnerabilidade. Isso se deve ao fato de ter sido utilizado na aplicação o *framework Hibernate*. Ao utilizá-lo, as consultas *SQL* passam por métodos internos que impedem que mais de uma operação seja repassada ao *driver* do banco de dados por uma única *string* de consulta.

Figura 5.31 – Resultado da varredura executada com o *Dirb*.



Fonte: Captura de tela realizada pelo autor.

Figura 5.32 – Vulnerabilidade encontrada pela ferramenta *OWASP ZAP*.



Fonte: Captura de tela realizada pelo autor.

Estágio 5. Exploração

A única vulnerabilidade encontrada na aplicação foi a vulnerabilidade de *Sql Injection*. Como a tentativa de ataque realizada com a aplicação *Sqlmap* não obteve êxito, aplicou-se téc-

Figura 5.33 – Varredura executada pela ferramenta *Nmap*.

```
Starting Nmap 7.80 ( https://nmap.org ) at 2019-11-04 11:02 -03
Nmap scan report for tenant-injection.herokuapp.com (34.249.148.208)
Host is up (0.24s latency).
Other addresses for tenant-injection.herokuapp.com (not scanned): 34.255.19.16 52.18.167.83 52.16.211.1
75 52.210.39.69 52.17.181.235 54.171.254.93 34.241.172.109
rDNS record for 34.249.148.208: ec2-34-249-148-208.eu-west-1.compute.amazonaws.com
Not shown: 65534 filtered ports
PORT    STATE SERVICE
443/tcp open  https
Warning: OSScan results may be unreliable because we could not find at least 1 open and 1 closed port
Device type: WAP|PBX
Running (JUST GUESSING): Comtrend embedded (87%), Vodavi embedded (87%), Gemtek embedded (85%), Siemens
 embedded (85%)
OS CPE: cpe:/h:comtrend:ct536 cpe:/h:vodavi:xts-ip cpe:/h:gemtek:p360 cpe:/h:siemens:gigaset_se515dsl
Aggressive OS guesses: Comtrend CT536 wireless ADSL router (87%), Vodavi XTS-IP PBX (87%), Gemtek P360
WAP or Siemens Gigaset SE515dsl wireless broadband router (85%)
No exact OS matches for host (test conditions non-ideal).

OS detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 593.09 seconds
```

Fonte: Captura de tela realizada pelo autor.

nicas manuais para verificar se a aplicação estaria vulnerável a *SQL Injection*, como preencher campos de inserção com códigos *SQL* com o intuito de confundir o interpretador. Inserir o código *'or'1'='1* nos campos de *login* e senha altera a consulta *SQL* de modo a fazer com que a resposta seja sempre verdadeira, possibilitando o acesso a aplicação sem conhecer o *login* ou senha. Na Figura 5.34 são expostos as entradas utilizadas no ataque e a autenticação alcançada.

Figura 5.34 – Ataque *SQL Injection*.

Fonte: Captura de tela realizada pelo autor.

Outros códigos foram inseridos manualmente nos campos de inserção de dados e na *URL* da aplicação, como *UNION* e códigos para tentar descobrir o nome da base de dados e tentar exibir alguns dados dos *tenants*, mas nenhuma outra tentativa foi bem sucedida.

Estágio 6. Pós-exploração

Nenhum ataque realizado na etapa anterior atingiu o objetivo de conseguir acesso aos dados dos *tenants*, pois não havia vulnerabilidade que atingisse esta camada da aplicação. Portanto, não foi executada a etapa Pós-exploração neste caso de teste de invasão.

Estágio 7. Relatórios

A seguir encontra-se um trecho de código da classe responsável pela autenticação dos usuários:

```
Query query = session.createSQLQuery("select * from user where
    ↪ username = '" + username + "' and password = '" +
    ↪ password + "' limit 0,1").addEntity(User.class);
```

Analisando o código acima é possível observar que a consulta *SQL* resultante deste método pode ser manipulada para confundir o interpretador da linguagem de consulta, o que caracteriza vulnerabilidade de *SQL Injection*. Por este motivo foi possível realizar autenticação na instância do *tenant* sem dispor das informações de usuário e senha. Nota-se, no entanto, que é utilizado o método *.createSQLQuery* do *Hibernate*. Este método impede que mais de uma consulta seja realizada na base de dados por meio da *string* de consulta gerada por este código. Por este motivo, tentativas de ataques que utilizaram a ferramenta *Sqlmap* e as outras tentativas manuais de ataque não foram bem sucedidas.

Entretanto, isso não protege completamente a aplicação da vulnerabilidade *SQL Injection*, pois ainda é possível alterar a consulta adicionando código nos campos de inserção de dados que não contenham o caracter ponto e vírgula (;). Contudo não se pode afirmar que outras aplicações desenvolvidas com o mesmo *framework* estão livres de exploração da vulnerabilidade de *SQL Injection*.

Apesar de não ser possível obter acesso aos dados dos *tenants*, foi possível executar ataques que permitem acessar a página de qualquer *tenant*. A utilização do *framework Hibernate* evitou que alguns tipos de ataques que exploram a vulnerabilidade *SQL Injection* fossem executados, fornecendo uma pequena proteção aos dados. Contudo, foi possível executar com sucesso um ataque de *SQL Injection* no qual foi possível realizar *login* sem o conhecimento do usuário e senha, o que poderia causar sérios danos a qualquer *tenant*.

5.5 Considerações finais

Nos três experimentos executados neste trabalho foi possível encontrar e explorar vulnerabilidades, evidenciando problemas graves que podem acarretar em muitos prejuízos para os provedores e utilizadores dos serviços.

No Experimento 1 detectou-se a vulnerabilidade *Cross-Site Scripting* (*XSS*) a partir da identificação de uma biblioteca utilizada na aplicação em uma versão vulnerável. A partir desta descoberta alguns códigos foram inseridos na aplicação, o que descartou a possibilidade da presença da vulnerabilidade *Cross-Site Request Forgery* (*CSRF*), mas ratificou a presença da vulnerabilidade *XSS*.

Códigos foram inseridos na página inicial de algumas instâncias de *tenants* para explorar a vulnerabilidade *XSS* e foi possível excluir o conteúdo destas. Um ataque como este em ambientes reais causaria muitos danos aos usuários e provedores do serviço, mas não são ataques que aproveitam das características da arquitetura *multi-tenant* para ampliar os ataques.

No Experimento 2 a vulnerabilidade *XSS* foi detectada da mesma maneira que no Experimento 1 e foi confirmada pela ferramenta *OWASP ZAP* e inserção de código. Além disso, também foi descoberto que não era utilizado na aplicação a *flag HttpOnly*, mecanismo que impede que o conteúdo do *cookie* seja acessado pela *API Javascript*. A não utilização deste mecanismo indica que a aplicação pode ser vulnerável a *CSRF*.

Em seguida foi inserida em uma página da aplicação um código **Javascript** que exibe, por meio de um alerta do navegador, o conteúdo do *cookie* da sessão. A exibição do conteúdo do *cookie* confirmou a presença da vulnerabilidade *CSRF*.

Para explorar a vulnerabilidade *CSRF* detectada na aplicação, iniciou-se, com a ferramenta *Burp Suite*, uma interceptação de requisições feitas pelo navegador. Em seguida realizou-se o início de uma sessão de administrador na instância do *tenant* 1 e foi feita a exclusão de um produto desta versão. A requisição capturada foi alterada e reenviada a aplicação *multi-tenant*, que confiou na requisição recebida por utilizar um *cookie* válido. A requisição foi executada, o que resultou na exclusão de um produto da instância do *tenant* 2.

Diferentemente do ataque realizado no Experimento 1, o ataque feito sobre a aplicação do Experimento 2 utilizou as características da arquitetura *multi-tenant* para ampliar o ataque, resultando na exclusão de um produto da instância de um cliente através da exploração da vulnerabilidade na instância de outro, evento que não poderia ser replicado em uma aplicação *web* convencional.

No Experimento 3 a vulnerabilidade de *SQL Injection* foi detectada pela ferramenta *OWASP ZAP*, mas a tentativa de exploração desta vulnerabilidade com a ferramenta *Sqlmap* não obteve êxito. Contudo várias tentativas "manuais"de exploração da vulnerabilidade foram realizadas e conseguiu-se iniciar sessão sem o conhecimento do usuário ou da senha.

Assim como no teste do Experimento 1, o ataque realizado no teste do Experimento 3 seria possível ser realizado em qualquer aplicação *web* convencional, pois não utilizou as características da arquitetura *multi-tenant* para ampliar o ataque. No entanto a falha detectada e explorada é de alto risco e deve ser corrigida.

Os experimentos são sumarizados no Quadro 5.2.

Tabela 5.2 – Quadro de experimentos.

Experimento	Aplicação	Persistência dos dados	Vulnerabilidades	Ataques obtiveram êxito?	Usufruiu das características da arquitetura?
1	iCardapio	BD compartilhados	XSS	Sim	Não
2	MtShop	BD isolados	CSRF	Sim	Sim
3	ToyExample	BD isolados	SQL Injection	Sim	Não

6 CONCLUSÃO E TRABALHOS FUTUROS

O objetivo deste trabalho foi identificar e explorar vulnerabilidades em sistemas baseados na arquitetura *multi-tenant*.

Neste trabalho, foram empregados diversos testes de invasão em três aplicações *multi-tenant* distintas. Foram descobertas vulnerabilidades em todas as aplicações. Todas as ferramentas utilizadas nos testes são ferramentas gratuitas e, algumas delas, *open source*.

Três tipos de vulnerabilidades foram exploradas: *Cross-site Script*, *Cross-site Request Forgery* e *SQL Injection*.

Ao explorar as vulnerabilidades foi possível alterar conteúdo das páginas da aplicação, alterar e acessar os dados da aplicação de qualquer *tenant* sem o conhecimento do *login* e senha.

Os testes de invasão empregados neste trabalho foram relativamente bem sucedidos. Foi possível causar danos nas diferentes aplicações testadas e foi possível mostrar como determinadas vulnerabilidades foram exploradas além de descrever por que não foi possível realizar alguns tipos de ataques. No entanto, não é possível garantir que as razões que impediram o sucesso de certos ataques sejam encontradas em qualquer aplicação *multi-tenant*.

Como um trabalho futuro pretende-se investigar o impacto de aumentar a carga para uma das instâncias fornecidas por uma aplicação *multi-tenant* e simultaneamente executar o pentest nas outras instâncias. A ideia é observar se vulnerabilidades podem ser identificadas e exploradas neste cenário.

As aplicações *multi-tenant* constituem grande parte do tipo de aplicações desenvolvidas atualmente. Os benefícios de aplicações que adotam esta arquitetura são muito atrativos para os desenvolvedores, *stakeholder* e clientes. Portanto, torna-se importante continuar realizando testes de invasão em aplicações *multi-tenant* para atingir um nível satisfatório de isolamento dos dados e que os desafios em relação a segurança da informação sejam bem compreendidas para se construir um ambiente de confiança entre os provedores dos serviços e os usuários.

REFERÊNCIAS

ARMBRUST, M. et al. A view of cloud computing. **Commun. ACM**, ACM, New York, NY, USA, v. 53, n. 4, p. 50–58, abr. 2010. ISSN 0001-0782. Disponível em: <http://doi.acm.org/10.1145/1721654.1721672>. Acesso em: 11/03/2019.

ASSUNçãO, M. F. **Segredos do Hacker Ético**. 5th. ed. [S.l.]: Visual Books, 2014. ISBN 9788575022856.

BACUDIO, A. G. et al. An overview of penetration testing. **International Journal of Network Security Its Applications**, v. 3, p. 19–38, 2011. Disponível em: <https://www.researchgate.net/publication/274174058_An_Overview_of_Penetration_Testing>. Acesso em: 31/03/2019.

BAI, X. et al. Cloud testing tools. in service oriented system engineering (sose). IEEE 6th International Symposium, p. 1–12, 2011. Disponível em: <https://www.researchgate.net/publication/220838758_Cloud_testing_tools>. Acesso em: 20/02/2019.

BASILI, V.; WEISS, D. A methodology for collecting valid software engineering data. **IEEE Trans. Software Eng.**, v. 10, p. 728–738, 1984. Disponível em: <https://www.researchgate.net/publication/220070466_A_Methodology_for_Collecting_Valid_Software_Engineering_Data>. Acesso em: 10/10/2019.

BASILI, V. R.; CALDIERA, G.; ROMBACH, H. D. The goal question metric approach. In: . [s.n.], 1994. Disponível em: <https://www.semanticscholar.org/paper/The-Goal-Question-Metric-Approach-Basili-Caldiera/02e65151786574852007ecd007ee270c50470af0>. Acesso em: 10/10/2019.

BASSO, T. **Uma Abordagem para Avaliação da Eficácia de Scanners de Vulnerabilidades em Aplicações Web**. 2010. Disponível em: <http://repositorio.unicamp.br/bitstream/REPOSIP/261545/1/Basso_Tania_M.pdf>. Acesso em: 30/04/2019.

BEZEMER, C.-P.; ZAIDMAN, A. Multi-tenant saas applications: Maintenance dream or nightmare? In: **Proceedings of the Joint ERCIM Workshop on Software Evolution (EVOL) and International Workshop on Principles of Software Evolution (IWPSE)**. ACM, 2010. (IWPSE-EVOL '10), p. 88–92. ISBN 978-1-4503-0128-2. Disponível em: <http://doi.acm.org/10.1145/1862372.1862393>. Acesso em: 11/03/2019.

BILBAO, B. **75Scripting Attacks**. 2014. Disponível em: <https://sensorstechforum.com/75-of-the-websites-on-weather-com-vulnerable-to-cross-site-scripting-attacks/>. Acesso em: 30/04/2019.

CERT. **ncidentes Reportados ao CERT.br – Janeiro a Dezembro de 2018**. 2018. Disponível em: <https://www.cert.br/stats/incidentes/2018-jan-dec/tipos-ataque.html>. Acesso em: 30/04/2019.

CHONG, F.; CARRARO, G. Architecture strategies for catching the long tail. **MSDN Library, Microsoft Corporation**, p. 9–10, 01 2006b. Disponível em: <http://cistratech.com/whitepapers/MS_longtailsaas.pdf>. Acesso em: 11/03/2019.

CHONG, F.; CARRARO, G.; WOLTER, R. **Multi-Tenant Data Architecture**. 2006a. Disponível em: <http://ramblingsofraju.com/wp-content/uploads/2016/08/Multi-Tenant-Data-Architecture.pdf>. Acesso em: 20/02/2019.

Cybersecurity Ventures. **Cybercrime Damages $6 Trillion By 2021**. 2016. Disponível em: <https://cybersecurityventures.com/hackerpocalypse-cybercrime-report-2016/>. Acesso em: 20/02/2019.

Dey, H.; Islam, R.; Arif, H. An integrated model to make cloud authentication and multi-tenancy more secure. In: **2019 International Conference on Robotics,Electrical and Signal Processing Techniques (ICREST)**. [S.l.: s.n.], 2019. p. 502–506.

GAO, J.; BAI, X.; TSAI, W.-T. Cloud testing-issues, challenges, needs and practice. Software Engineering:An International Journal, p. 9–23, 2011. Disponível em: <https://www.researchgate.net/publication/285328751_Cloud_testing-issues_challenges_needs_and_practice>. Acesso em: 20/02/2019.

GREGG, M. **Certified Ethical Hacker Exam Prep (Exam Prep 2 (Que Publishing))**. Indianapolis, IN, USA: Que Corp., 2006. ISBN 0789735318.

HAWEDI, M.; TALHI, C.; BOUCHENEB, H. Multi-tenant intrusion detection system for public cloud (mtids). **The Journal of Supercomputing**, v. 74, 2018. Disponível em: <https://www.researchgate.net/publication/327383907_Multi-tenant_intrusion_detection_system_for_public_cloud_MTIDS>. Acesso em: 10/10/2019.

HOSNI, A. An analysis of cloud computing multitenancy security challenges. **International Journal of Advanced Research**, v. 5, p. 850–855, 09 2017.

IEEE Computer Society. **IEEE CS2022 Report**. 2014. Disponível em: <https://ieeecs-media.computer.org/assets/pdf/2022Report.pdf>. Acesso em: 27/05/2019.

JANSEN, S.; HOUBEN, G.-J.; BRINKKEMPER, S. Customization realization in multi-tenant web applications: Case studies from the library sector. In: . [s.n.], 2010. v. 6189, p. 445–459. Disponível em: <https://www.researchgate.net/publication/220940366_Customization_Realization_in_Multi-tenant_Web_Applications_Case_Studies_from_the_Library_Sector>. Acesso em: 31/03/2019.

JUNG, C. **Metodologia para Pesquisa Desenvolvimento: Aplicada a Novas Tecnologias, Produtos e Processos**. [s.n.], 2004. Disponível em: <https://www.researchgate.net/publication/216808015_Metodologia_para_Pesquisa_Desenvolvimento_Aplicada_a_Novas_Tecnologias_Produtos_e_Processos>. Acesso em: 20/02/2019.

KABBEDIJK, J. et al. Defining multi-tenancy: A systematic mapping study on the academic and the industrial perspective. **Journal of Systems and Software**, v. 100, n. 0, p. 139 – 148, 2015. ISSN 0164-1212. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0164121214002313>.

KEMP, S. **Digital in 2018: World's internet users pass the 4 billion mark**. 2018. Disponível em: <https://wearesocial.com/uk/blog/2018/01/global-digital-report-2018>. Acesso em: 20/02/2019.

KWOK, T.; MOHINDRA, A. Resource calculations with constraints, and placement of tenants and instances for multi-tenant saas applications. p. 633–648, 2008. Disponível em: <https://www.researchgate.net/publication/221050769_Resource_Calculations_with_Constraints_and_Placement_of_Tenants_and_Instances_for_Multi-tenant_SaaS_Applications>. Acesso em: 11/03/2019.

KWOK, T.; NGUYEN, T.; LAM, L. A software as a service with multi-tenancy support for an electronic contract management application. **IEEE International Conference on Services Computing**, v. 2, p. 179–186, 2008. Disponível em: <https://ieeexplore.ieee.org/document/4578523>. Acesso em: 20/02/2019.

LABARGE, R.; MCGUIRE, T. Cloud penetration testing. **International Journal on Cloud Computing: Services and Architecture**, v. 2, 2013. Disponível em: <https://www.researchgate.net/publication/234083172_Cloud_Penetration_Testing>. Acesso em: 10/10/2019.

MANDUCA, A. M. et al. A nonintrusive approach for implementing single database, multitenant services from web applications. In: **29th ACM Symposium on Applied Computing**. [S.l.: s.n.], 2014. p. 751–756. Acesso em: 10/10/2019.

MäKILä, T. et al. How to define software-as-a-service – an empirical study of finnish saas providers. Tyrväinen P., Jansen S., Cusumano M.A. (eds) Software Business, p. 115–116, 2010. Disponível em: <https://link.springer.com/chapter/10.1007/978-3-642-13633-7_10>.

MüLLER, J. et al. Customizing enterprise software as a service applications: Back-end extension in a multi-tenancy environment. In: . [s.n.], 2009. p. 66–77. Disponível em: <https://www.researchgate.net/publication/220712041_Customizing_Enterprise_Software_as_a_Service_Applications_Back-End_Extension_in_a_Multi-tenancy_Environment>. Acesso em: 31/03/2019.

National Institute of Standards and Technology (NIST). **NIST Cloud Computing Program - NCCP**. 2010. Disponível em: <https://www.nist.gov/programs-projects/nist-cloud-computing-program-nccp>. Acesso em: 20/02/2019.

NITU. Configurability in saas (software as a service) applications. In: . [s.n.], 2009. p. 19–26. Disponível em: <https://www.researchgate.net/publication/220796433_Configurability_in_SaaS_software_as_a_service_applications>. Acesso em: 31/03/2019.

OWASP. Owasp top ten - 2017. In: . [s.n.], 2017. Disponível em: <https://www.owasp.org/images/7/72/OWASP_Top_10-2017_%28en%29.pdf.pdf>. Acesso em: 30/04/2019.

PALIWAL, S. **Cloud application services (SaaS) – Multi-Tenant Data Architecture**. 2012. Disponível em: <https://www.cmg.org/wp-content/uploads/2012/11/m_94_4.pdf>. Acesso em: 11/03/2019.

PINTO, H. S. C. et al. A systematic mapping study on the multi-tenant architecture of saas systems. In: . [s.n.], 2016. Disponível em: <https://www.researchgate.net/publication/306255127_A_Systematic_Mapping_Study_on_the_Multi-tenant_Architecture_of_SaaS_Systems>. Acesso em: 27/05/2019.

PINTO, V. H. S. C. **Teste de Software em Aplicações na Nuvem: Evidências empíricas sobre a efetividade, custo e aplicabilidade**. Tese (Doutorado), 2016.

PINTO, V. H. S. C.; SOUZA, S. R. S. A preliminary fault taxonomy for multi-tenant saas systems. In: **2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)**. [S.l.: s.n.], 2019. p. 178–187. ISSN null.

PINTO, V. H. S. C.; SOUZA, S. R. S.; SOUZA, P. S. L. A Preliminary Fault Taxonomy for Multi-tenant SaaS systems. In: **19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid 2019)**. [s.n.], 2019a. p. 178–187. Disponível em: <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8752873>. Acesso em: 10/10/2019.

RU, J.; GRUNDY, J.; KEUNG, J. Software engineering for multi-tenancy computing challenges and implications. International Workshop on Innovative Software Development Methodologies and Practices, InnoSWDev 2014 - Proceedings, p. 1–10, 2014. Disponível em: <https://www.researchgate.net/publication/289750038_Software_engineering_for_multi-tenancy_computing_challenges_and_implications>. Acesso em: 20/02/2019.

RU, J.; KEUNG, J. An empirical investigation on the simulation of priority and shortest-job-first scheduling for cloud-based software systems. IEEE Computer Society, Washington, DC, USA, p. 78–87, 2013. Disponível em: <http://dx.doi.org/10.1109/ASWEC.2013.19>.

Salesforce. The force.com multitenant architecture understanding the design of salesforce.com's internet application development platform. In: . [s.n.], 2008. Disponível em: <https://pdfs.semanticscholar.org/e145/56cdc11c4c911597cbca7240e6939f48a733.pdf?_ga=2.86690648.1721360409.1553282872-1526417649.1553282872>. Acesso em: 11/03/2019.

SecTools. Sectools.org: Top 125 network security tools. In: . [s.n.], 2019. Disponível em: <https://sectools.org/>. Acesso em: 10/10/2019.

SILVA, E. D.; MENEZES, E. M. **Metodologia da Pesquisa e Elaboração de Dissertação**. [s.n.], 2005. Disponível em: <https://www.researchgate.net/publication/312125489_Metodologia_da_Pesquisa_e_Elaboracao_de_Dissertacao>. Acesso em: 20/02/2019.

SILVA, R. Ronner Tertulino da et al. InvestigaÇÃo de seguranÇa no moodle. **RENOTE**, v. 12, 2015. Disponível em: <https://seer.ufrgs.br/renote/article/view/53501/33018>. Acesso em: 30/04/2019.

SOLóRZANO, A. L. V.; CHARãO, A. S. Explorando a plataforma de computação em nuvem heroku para execução de programas paralelos com openmp. In: **Anais da XVII Escola Regional de Alto Desempenho do Estado do Rio Grande do Sul**. SBC, 2017. Disponível em: <https://sol.sbc.org.br/index.php/eradrs/article/view/2962>. Acesso em: 10/10/2019.

Song, H.; Chauvel, F.; Solberg, A. Deep customization of multi-tenant saas using intrusive microservices. In: **2018 IEEE/ACM 40th International Conference on Software Engineering: New Ideas and Emerging Technologies Results (ICSE-NIER)**. [S.l.: s.n.], 2018. p. 97–100.

Song, H. et al. Customizing multi-tenant saas by microservices: A reference architecture. In: **2019 IEEE International Conference on Web Services (ICWS)**. [S.l.: s.n.], 2019. p. 446–448.

TORRES, A. F. F. **Os referenciais de segurança da informação e a melhoria contínua: um caso exploratório**. 2014. Disponível em: <http://recipp.ipp.pt/bitstream/10400.22/5588/1/DM_AndreTorres_2014_MEI.pdf>. Acesso em: 30/04/2019.

TURBAN, E. et al. **Business Intelligence: Um enfoque gerencial para a inteligência do negócio**. 1. ed. [S.l.]: Bookman, 2009. 21 p.

VAQUERO, L. M. et al. A break in the clouds: Towards a cloud definition. **SIGCOMM Comput. Commun. Rev.**, ACM, New York, NY, USA, v. 39, n. 1, p. 50–55, dez. 2008. ISSN 0146-4833. Disponível em: <http://doi.acm.org/10.1145/1496091.1496100>. Acesso em: 20/02/2019.

VASHISTHA, A.; AHMED, P. Saas multi-tenancy isolation testing challenges and issues. International Journal of Soft Computing and Engineering (IJSCE), 2012. Disponível em: <https://pdfs.semanticscholar.org/4457/1e8644e093442c152b54742a1e1fb4f37f4e.pdf>. Acesso em: 20/02/2019.

WANG, Z. H. et al. A study and performance evaluation of the multi-tenant data tier design patterns for service oriented computing. **E-Business Engineering, IEEE International Conference on**, p. 94–101, 2008. Disponível em: <https://www.researchgate.net/publication/232638382_A_Study_and_Performance_Evaluation_of_the_Multi-Tenant_Data_Tier_Design_Patterns_for_Service_Oriented_Computing>. Acesso em: 31/03/2019.

WARFIELD, B. **Multitenancy Can Have a 16:1 Cost Advantage Over Single-Tenant**. 2007. Disponível em: <https://smoothspan.com/2007/10/28/multitenancy-can-have-a-161-cost-advantage-over-single-tenant/>. Acesso em: 31/03/2019.

WEIDMAN, G. **Penetration Testing: A Hands-On Introduction to Hacking**. 1st. ed. San Francisco, CA, USA: No Starch Press, 2014. 1 p. ISBN 1593275641, 9781593275648.

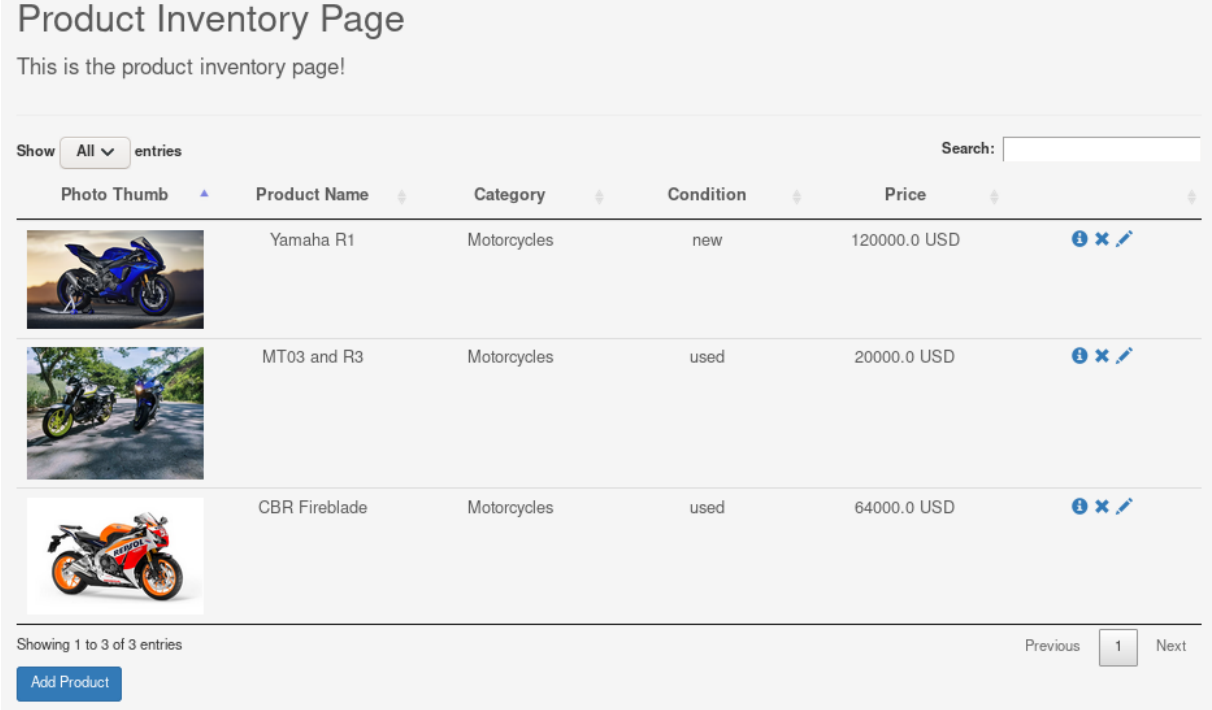
ZISSIS, D.; LEKKAS, D. Addressing cloud computing security issues. **Future Generation Comp. Syst.**, v. 28, 2012. Disponível em: <https://www.researchgate.net/publication/220285301_Addressing_cloud_computing_security_issues>. Acesso em: 10/10/2019.

APÊNDICE A – Outras funcionalidades da aplicação *MtShop*

Clicar na opção *Logout* faz com que a sessão seja encerrada e o usuário é redirecionado para a página de login.

Clicar na opção *Product Inventory* exibe uma página muito parecida com a página *Products*, porém contém um botão *Add Product* na parte inferior da página, e três botões em cada item cadastrado. Figura 1.

Figura 1 – Página exibida ao clicar no botão *Product Inventory*.



Fonte: Captura de tela realizada pelo autor.

O botão *Add Product* direciona o usuário para uma página com formulários para inserção de informações sobre o produto a ser cadastrado, além de um botão no qual é possível enviar a imagem do produto.

Os botões em cada produto do catálogo, da esquerda para a direita, possuem as seguintes funções: o primeiro exibe as informações do produto; o segundo exclui o produto do catálogo; e o terceiro exibe uma tela na qual é possível alterar as informações do produto.

Quando é iniciada uma sessão com um usuário que não é administrador, as operações disponíveis são outras. Ao entrar na página *Products* e clicar na opção que exibe os detalhes sobre um produto, três botões são exibidos para o usuário: *Back*, *Order Now* e *View Cart*.

O botão *Back* direciona o usuário de volta para a página *Products*. O botão *Order Now* adiciona o produto ao carrinho do usuário. A opção *View Cart* exibe uma página que contem uma lista de todos os produtos adicionados ao carrinho do usuário

As opções da página de carrinho do cliente são: remover o produto do carrinho (botão *remove*); limpar o carrinho (botão *Clear cart*); continuar comprando (botão *Continue Shopping*); e finalizar compra (botão *Check out*). Esta última opção redireciona para uma página na qual são exibidas algumas informações do pedido e endereços de cobrança e entrega.

Os botões *Cancel* e *Back* da página de pedidos voltam para a página do carrinho do cliente. O botão *Submit Order* finaliza a compra e exibe ao usuário a seguinte mensagem: “Thank you for your business! Your order will be shipped in two business days!”.

Ao iniciar uma sessão com um usuário cliente, um botão para acessar o carrinho também é adicionado a barra de menu da aplicação.