MESTRADO EM ENGENHARIA DE TECNOLOGIAS E

SISTEMAS WEB

Sistema de Monitorização de Segurança de Containers

Aluno: Vladimir Kaznacheev

Orientador: Professor Doutor Domingos Martinho

DISSERTAÇÃO

VILA NOVA DE GAIA

SETEMBRO | 2022

Tese de Mestrado realizada sob a orientação do Prof. Doutor Domingos Martinho, apresentada ao ISLA Gaia para obtenção do grau de Mestre em Engenharia de Tecnologias e Sistemas Web, conforme o despacho nº 9371/2020, publicado no nº 191, na 2ª Série do Diário da República, em 30 de setembro de 2002.

Agradecimentos

Gostaria de expressar os agradecimentos ao Professor Doutor Domingos Martinho pela ajuda infinita que me deu ao longo do desenvolvimento da dissertação, bem como pela paciência, melhorias e correções que foram criticamente necessários para o sucesso do trabalho final.

Queria expressar agradecimentos ao direção do curso e aos melhores professores do ISLA Gaia e ISLA Santarém pelo apoio e sugestões de formas racionais e eficazes para resolver dúvidas e problemas ao longo do curso e durante desenvolvimento da dissertação.

Agradecimentos imensos ao meu amigo Professor André Costa sem cuja ajuda, explicações, apoio e os conhecimentos que me foram transferidos, a minha defesa jamais teria acontecido.

Um lugar especial na minha longa lista de agradecimentos vai para a minha esposa Elena, que magicamente me energizou tão necessária para mim.

Resumo

Este trabalho teve como objetivo final monitorizar os parâmetros de segurança e o seu estado em ambientes de container construídos em diversas tecnologias. Pretendeu-se, igualmente, fornecer aos utilizadores do sistema não só informações abrangentes acerca dos containers em execução, mas também garantir a segurança dos dados e da infraestrutura.
O sistema desenvolvido basei-as a recursos da web. A linguagem de programação Python, nomeadamente o Django Framework, foi utilizada como backend para a criação de aplicações web e para lógica computacional. Para o desenvolvimento do frontend e a GUI de utilizadores utilizou-se a linguagem de programação Java Script, ou seja, o React Framework. Optou-se pelo Scrum como metodologia de desenvolvimento. O desenvolvimento foi realizado de forma modular com sprints incluindo desenvolvimento de requisitos, desenvolvimento de recursos, testes e correções de bugs.
Como resultado da pesquisa, foram solucionados os problemas de avaliação da segurança da infraestrutura containerizada e do fornecedor cruzando as tecnologias utilizadas para sua construção. Além disso, funções e visualizações foram implementadas apresentando-se recomendações para aumentar o nível de segurança, fechando vulnerabilidades conhecidas e reduzindo a superfície de um possível ataque. Como resultado do estudo, foram confirmados os problemas existentes na segurança das infraestruturas de contentores, comuns a todos os fornecedores e específicos a cada uma das tecnologias de contentorização. O estudo mostrou que o monitorização de segurança em tempo real é mais eficaz para trazer a segurança dos sistemas de informação a um nível aceitável do que o monitorização periódico e o uso das configurações fornecidas.

Abstract

The final objective of this work was to monitor security parameters and their status in container environments built on different technologies. It was also intended to provide system users not only comprehensive information about running containers, but also to ensure data and infrastructure security. The developed system based them on web resources. The Python programming language, namely the Django Framework, was used as a backend for creating web applications and for computational logic. For the development of the frontend and the user GUI, the Java Script programming language was used, that is, the React Framework. Scrum was chosen as a development methodology. Development was carried out in a modular fashion with sprints including requirements development, feature development, testing and bug fixes.
As a result of the research, the problems of evaluating the security of the containerized infrastructure and the supplier were solved by crossing the technologies used for its construction. In addition, functions and views were implemented presenting recommendations to increase the security level, closing known vulnerabilities and reducing the surface of a possible attack. Because of the study, the existing problems in the security of container infrastructures, common to all suppliers and specific to each of the containerization technologies, were confirmed. The study showed that real-time security monitoring is more effective in bringing the security of information systems to an acceptable level than periodic monitoring and use of the provided settings.

Keywords: cybersecurity; container; docker; monitoring.

INSTITUTO POLITÉCNICO DE GESTÃO E TECNOLOGIA

Sistema de Monitorização de Segurança de Containers

Vladimir Kaznacheev

Aprovado em …../…./……..

Composição do Júri

_____Professor Doutor Firmino da Silva_____

Presidente

_____Professor Doutor Ricardo Vardasca_____

Arguente

_____Professor Doutor Domingos Martinho_____

Orientador

Vila Nova de Gaia

2022

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABELAS

INTRODUÇÃO

A cibersegurança é uma área das TIC que é responsável por proteger os ativos de informação por meio do tratamento de ameaças que comprometem as informações que são processadas, armazenadas e transportadas por sistemas de informação interligados. A segurança cibernética pode ser alcançada por meio do desenvolvimento sistemático, mas os desenvolvedores precisam estar cientes desses riscos e problemas de segurança. Se a segurança cibernética for fornecida após a conclusão do desenvolvimento, os sistemas serão construídos de forma insegura com bugs difíceis de corrigir (Díaz & Pérez, 2019).

De acordo com investigação da Markets and Markets, o mercado global de proteção ambiental em containers crescerá de US$ 568 milhões (2019) para US$ 2.178 milhões até 2024 a uma taxa de crescimento anual composta (CAGR) de 30,9%. Espera-se que o mercado global de segurança para ambientes de containerização tenha um potencial de crescimento significativo devido a um aumento de vulnerabilidades e ciberataques, um grande número de fornecedores de plataformas de código aberto, a crescente popularidade de microserviços, transformação digital progressiva entre empresas e a necessidade de aderir a regulamentações políticas.

Para manter uma rede segura, as organizações estão instituindo medidas preventivas com a esperança de prevenir e impedir ataques cibernéticos. As organizações estão contratando uma infinidade de especialistas em segurança cibernética que foram treinados e certificados nas áreas de segurança de computadores e redes. De acordo com a Aqua Security, uma empresa de tecnologia com sede em Massachusetts, “a equipe de pesquisa cibernética da Aqua Security tem visto ataques cada vez mais sofisticados em containers – ameaças que escaparam da deteção por scanners estáticos. Esses ataques de malware incluem mineração de criptomoedas, roubo de credenciais, exfiltração de dados ou ataques DDoS – ameaças que só são detectáveis em containers em execução. Para se proteger contra esse malware furtivo, a solução de sandbox de análise de container da Aqua, Aqua Dynamic Threat Analysis (DTA), detecta dinamicamente os riscos ocultos nas imagens do container, imitando a superfície da ameaça como se as imagens estivessem sendo executadas em produção. Ele encontra comportamentos maliciosos que só podem ser observados quando uma imagem é executada como um container (Dennis Jerome, 2021).

À medida que os investimentos se expandem e os serviços em nuvem são integrados a todos os aspectos da vida, surgem preocupações em torno da deteção de problemas de segurança tanto de praticantes quanto de acadêmicos. Enfatizando essas preocupações, a Alert Logic, uma empresa de segurança como serviço de big data, publicou um relatório, em 2015, onde afirma que “empresas que usam ambientes em nuvem são consideradas jackpot para hackers" (Thomas Watts, 2019).

Atualmente estão disponíveis várias ferramentas de verificação de containers (por exemplo, Clair, Anchore e Microscanner) que podem analisar imagens oficiais e da comunidade hospedadas no Docker Hub. A abordagem utilizada por essas ferramentas é que elas recolhem informações de pacotes (por exemplo, pacotes básicos do OS3) e as comparam com um banco de dados de vulnerabilidades.

O Qualys Vulnerability Management (VM) é um serviço baseado em nuvem que fornece uma visibilidade global de como os sistemas de TI são vulneráveis às ameaças mais recente. Ele identifica continuamente as ameaças e monitoriza as mudanças inesperadas na rede. É uma solução avançada, extensível e escalável para gerenciamento e análise contínua de vulnerabilidades. Este serviço é bem conhecido pela sua rápida implantação e integração com outros sistemas corporativos. (Ahtisham Hashmi, 2018). Os principais recursos do Qualys VM são:

- Monitorização e alertas constantes. As equipes de Segurança da Informação são alertadas sobre as possíveis ameaças quando a VM é integrada na Monitorização Contínua (CM), para que os problemas possam ser abordados antes que comecem a violar, agindo como sentinela para a nuvem.
- Deteção baseada em agente. Para estender a cobertura da rede aos ativos que não podem ser verificados, a VM trabalha com os Qualys Cloud Agents. Esses agentes autoatualizáveis e leves residem nos ativos que monitoram e não precisam de alterações de firewall, credenciais e janela de verificação.
- Cobertura e visibilidade abrangentes. Nos *end points* móveis e na nuvem, a identificação contínua de vulnerabilidades com precisão Six Sigma (99,99966%) e a proteção de ativos de TI no local são feitas pela Qualys VM. Incluindo a documentação de segurança automática para auditores de conformidade, o Qualys VM também gera relatórios baseados em função para várias partes interessadas.

O software externo NeuVector é utiliza para gerenciar a segurança do container de ciclo de vida completo. No NeuVector, as políticas podem ser definidas para controlar a comunicação no cluster, tornando obsoleta a necessidade de políticas de rede. Em ambos os casos, circunstâncias especiais impossibilitam a deteção de verdadeiros positivos para a verificação correspondente. Nessas situações, a verificação deve ser desabilitada para o projeto. (Müller, 2020).

A McAfee tem vindo a concentrar-se em quatro abordagens de engenharia para automatizar todo o ciclo de vida da defesa contra ameaças – abordagens centradas em parceria, baseadas em plataforma, experiências reinventadas e centradas na nuvem. Todas essas quatro abordagens são integradas em uma única plataforma para aproveitar os benefícios de cada uma. Consideramos que a abordagem baseada em plataforma é o núcleo estratégia de engenharia. A plataforma de orquestração foi projetada para automatizar várias atividades de ameaças ciclo de vida de defesa. Essa revisão nos permitiu propor uma taxonomia de orquestração de segurança para oferecer suporte a uma comparação e análise sistemáticas das soluções de orquestração de segurança existentes, conforme descrito. A taxonomia proposta consiste em várias dimensões e subdimensões para classificar as técnicas de orquestração de segurança. (Chadni Islam, 2020).

O processo de descobrimento de vulnerabilidades está envolvido tanto no meio corporativo privado quanto em organização publicas ou sem fins lucrativos. Diversas empresas se segurança da informação se encontram no mercado na atualidade, e oferecem dos mais variados serviços desde testes de penetração até implementação de protocolos ou serviços. Como é o caso da Trend Micro [Trend Micro, 2021] que oferece produtos e serviços e da subdivisão da Google, a qual busca por vulnerabilidades nos produtos utilizados e desenvolvidos na empresa. (Nikolas Jensen, 2021).

Open Source Software (OSS) é atualmente usado e integrado na maioria dos produtos comerciais. No entanto, a seleção de projetos de OSS para integração não é um processo simples, principalmente pela falta de modelos claros de seleção e falta de informações dos portais de OSS. A Black Duck, já estava desenvolvendo um produto para auditoria de OSS, com foco particular na análise da compatibilidade de licenças, e integrou funcionalidade com seus produtos. Em 2014, Ohloh tornou-se "Black Duck Open Hub". Finalmente, a Synopsys adquiriu a Black Duck e renomeou o Black Duck Open Hub para “Synopsys Open Hub”. Sinopse Open Hub é atualmente o único agregador OSS continuamente atualizado que inclui informações de diferentes projetos OSS de diferentes fontes (sistemas de controle

de versão, sistemas de rastreamento de problemas, bancos de dados de vulnerabilidades). Em janeiro de 2021, o OpenHub indexou quase 500 mil projetos e mais de 30 bilhões de linhas de código. Ele oferece flexibilidade para os utilizadores selecionarem as métricas para comparar estatísticas de projetos, idiomas, repositórios, etc. No entanto, faltam as facilidades de avaliação do OSS que permitem ajustar a importância das métricas selecionadas de acordo com as necessidades dos utilizadores para pontuar automaticamente o software candidato. Além disso, carece de informações relacionadas à popularidade da comunidade, documentação, disponibilidade de perguntas e respostas e outras informações. (Xiaozhou Li, 2022).

Com o novo desenvolvimento das tecnologias de comunicação da informação (TICs), como veículos autônomos, drones e inteligência artificial, a vida moderna está se tornando cada vez mais conveniente em quase todas as áreas da vida. As TIC não são adotadas apenas para melhorar a qualidade de vida; os campos da indústria também estão tentando introduzir novas TICs com a Indústria 4.0. De acordo com o relatório de ameaças da THALES, uma empresa de segurança global, 63% dos entrevistados entre mais de 1.100 executivos de segurança sênior descreveram que suas organizações aplicam novas tecnologias sem considerar os níveis de segurança. O relatório mostra que o desenvolvimento de TIC sem considerar a tecnologia de segurança pode causar incidentes de segurança irreversíveis. Portanto, precisamos considerar a segurança para ambientes de TIC mais seguros (Jun-Young Park, 2020). A Thales, com base em Paris, é uma grande empresa de eletrônicos no ramo de fornecimento de infraestrutura e segurança de transações para TI e serviços relacionados. ThalesCrypt, para proteger e proteger seus feeds de satélite. A ThalesCrypt oferece uma solução modular de ponta a ponta que inclui um descodificador, um recetor/descodificador e software de gerenciamento de direitos (Flournoy, 2021).

Um exemplo de falha na regra de política de segurança seria um pipeline de compilação com falha se houver problemas de vulnerabilidade e conformidade (VC) altos e críticos que já tenham uma correção. Diferentes políticas podem ser aplicadas com base na criticidade da infraestrutura, por exemplo, o ambiente de desenvolvimento tem uma política de segurança diferente do ambiente de produção. O mecanismo Anchore é um exemplo de ferramenta de código aberto que pode ser integrado à popular ferramenta de CD, Jenkins. Do lado comercial, o Prisma Cloud da Palo-Alto, entre outros, também pode ser utilizar para alcançar essa implementação e a verificação de vulnerabilidade e conformidade (Raheem, 2021).

O Kubernetes é uma das ferramentas mais utilizadas para gerenciar containers. Os resultados de uma pesquisa realizada pela StackRox em 2020 mostraram que o Kubernetes foi amplamente adotado por 91%, especialmente em ambientes de produção em até 75%. Além do uso crescente do Kubernetes para gerenciamento de containers, o StackRox também descobriu que 90% dos entrevistados incidentes de segurança experientes no Kubernetes e no ambiente containers nos últimos 12 meses. StackRox é uma empresa e inovadora que trabalha em containers e Kubernetes-Native Security. O Kubernetes conseguiu deslocar significativamente outras ferramentas de gerenciamento do mercado, como Docker Swarm e Amazon Elastic Container Service (ECS) (Ikhwan, 2021).

A investigação apresentada pode ser importante do ponto de vista do fato de que as novas tecnologias populares de containerização, diferindo na conveniência e velocidade de implementação, gestão de versões e portabilidade, que inevitavelmente apresentam problemas de segurança, e esta investigação pretende dar mais confiança às empresas, aos programadores e aos responsáveis pela gestão da segurança de que a segurança da sua infraestrutura e dados estão sob controle.

1.1 Motivação

Depois de analisar as tendências dos últimos anos no mundo digital, torna-se óbvio em empresas de grande e médio e pequena dimensão que a virtualização baseada em container, juntamente com a importância crítica da segurança cibernética e de dados, continuarão sendo os principais problemas no futuro próximo. Em particular, cheguei a essas conclusões observando um aumento acentuado no número de solicitações de clientes da minha empresa (Confident ltd.) para proteger a sua infraestrutura de containers. A questão é que os meios tradicionais de garantir a segurança cibernética e a segurança dos dados, como sistemas de prevenção de intrusão, firewall e DLP em uma infraestrutura de container nem sempre satisfazem a alta administração das empresas e os diretores de segurança num nível aceitável de segurança.

Muitos dos princípios e práticas da metodologia DevOps podem ser aplicados à segurança por meio da implementação de práticas DevSecOps, criando segurança desde o início (deslocando a segurança para a esquerda, automatizando testes e monitoramento de segurança e colaboração efetiva entre desenvolvedores e engenheiros de segurança (Díaz & Pérez, 2019).

DevSecOps é uma técnica para integrar princípios de segurança em um pipeline de integração contínua, entrega contínua e implantação contínua. Perceber os valores do DevOps em segurança de software significa que a revisão de segurança se torna uma parte ativa e integral do processo de desenvolvimento.

Dada a crescente procura por competências em segurança de containers no meu atual ambiente de trabalho, bem como em outras empresas e no mercado de trabalho em geral, decidi pesquisar os principais temas nesta área. O principal objetivo deste trabalho consiste em estudar as principais ameaças e vetores de possíveis ataques a infraestruturas de contentores, determinar os requisitos para o sistema de monitorização e detetar violações do nível aceitável da política de segurança.

1.2 Problema de investigação

O principal problema que se pretende resolver é automatizar a recolha e análise de tantos dados de segurança em tempo real quanto possível de containers de várias tecnologias. Portanto, esse problema deve ser dividido em vários subproblemas:

- identificação de fontes para coleta de dados;
- escolha de tecnologias, protocolos, métodos e viabilidade de armazenamento e acumulação de dados;
- seleção de esquemas, algoritmos e marcadores para a análise do comportamento anômalo de containers;
- realizar um cálculo do nível de segurança com base na análise realizada;
- preparação de recomendações para melhorar a sustentabilidade da infraestrutura de containers.

Além disso, no decorrer da pesquisa, foi desenvolvido um protótipo do sistema de monitorização. Para desenvolver este sistema, escolheu-se a seguinte stack de tecnologias:

- para frontend foi utilizado HTML, Java Script, CSS (React Framework);
- para backend foi utilizado Python (Django Framework).

1.3 Objetivos, geral e específicos

Para atingir as metas estabelecidas e solucionar os problemas anteriormente enunciados, definiram-se os seguintes objetivos especifícos:

- responder à pergunta sobre a possibilidade de substituir sistemas de segurança dispendiosos pelo sistema resultante do nosso próprio desenvolvimento;
- realizar um estudo de trabalhos de pesquisa anteriores sobre as questões de segurança da infraestrutura de containers;
- estudar possíveis métodos e esquemas de recolha de informações sobre containers e conexões entre containers;
- complementar o conhecimento sobre ameaças e vetores de possíveis ataques;
- definir uma lista de requisitos para a implementação de sensores de segurança;
- complementar o conhecimento existente sobre React e Django para implementar o backend e o frontend de um protótipo de sistema de monitorização de segurança de container;
- avaliar o nível atual de segurança e propor medidas para melhorá-lo.

1.4 Estrutura do relatório

No primeiro capítulo faz-se uma introdução da temática em estudo e apresentam-se os desenvolvimentos mais relevantes da industria. Este capítulo inclui ainda a apresentação da motivação para o desenvolvimento do trabalho e concluiu-se com a apresentação do objetivo geral e dos objetivos específicos.

No segundo capítulo apresenta-se o estado da arte incluindo as caraterísticas mais relevantes do ponto de vista tecnológico e dos estudos técnicos e académicos publicados. Neste capítulo apresentam-se ainda as tecnologias e ferramentas utilizadas para o desenvolvimento do protótipo desenvolvido.

No terceiro capítulo faz-se a apresentação da metodologia para o desenvolvimento do projeto, bem como as ações desenvolvidas para preparar o ambiente adequado ao desenvolvimento do protótipo.

No capítulo 4 faz-se a descrição detalhada dos passos seguidos para o desenvolvimento do protótipo apresentando-se em cada passo os resultados mais relevantes.

No capítulo 5 discutem-se os resultados obtidos quer na perspetiva teórica quer na perspetiva da aplicação desses resultados a contextos concretos. Ainda neste capítulo apresentam-se algumas das limitações do trabalho realizado.

Por último, apresentam-se as conclusões mias relevantes e concluiu-se com a apresentação dos caminhos que podem ser seguidos para eventuais desenvolvimentos da investigação.

REVISÃO DA LITERATURA

2.1 O que é e para que serve a virtualização baseada em container

A virtualização baseada em container surgiu como uma alternativa leve para VMs. Muitos containers podem compartilhar o mesmo kernel do sistema operativo em vez de ter uma A virtualização é um conceito antigo, que vem sendo utilizado na computação em nuvem, após o IaaS ter sido aceito como uma técnica crucial para constituição de sistemas, provisionamento de recursos e multilocação. A Figura 1 mostra a arquitetura da máquina virtual e a arquitetura dos containers. O hipervisor está entre os sistemas operacionais host e convidado. É uma plataforma virtual e lida com mais de um sistema operativo no servidor e que funciona entre o sistema operativo e a CPU. Os containers utilizados como um método de virtualização no nível do sistema operativo. A Figura 1 mostra que em um host de controle único existem muitos containers, que são isolados. Recursos como Rede, Memória, CPU e Block I/O são alocados pelo kernel host OS e também lida com cgroups sem iniciar a máquina de virtualização cópia dedicada para cada um, como nas VMs (Babak Bashari Rad, 2017).

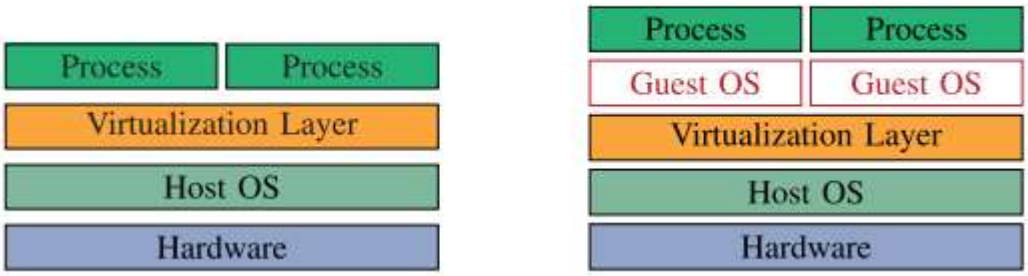

Figura 1. Modelos de virtualização baseado em hipervisor clássica e containers.

Isso reduz bastante o tempo de inicialização e os recursos necessários para cada imagem. Por exemplo, um container pode iniciar em 50 milissegundos, enquanto uma VM pode levar de 30 a 40 segundos para iniciar. Muitas tecnologias de container estão disponíveis, como LXC, OpenVZ, Linux-Vserver, Docker. Os containers são uma opção mais plausível para microserviços do que as VMs devido aos inúmeros benefícios, como ser leve, rápido, fácil de implantar e permitir melhor utilização de recursos e controle de versão. Os containers estão sendo usados para diferentes aplicações, como serviços de Internet das Coisas (IoT), carros inteligentes, computação na nuvem, redes de serviço e assim por diante.

A introdução de arquiteturas de microserviços ajudou a aumentar a agilidade do software, em que as partes do software tornaram-se unidades independentes de desenvolvimento, controle de versão, implantação e escalonamento. Os microserviços são usados por várias organizações, como Amazon, Spotify, Netflix e Twitter para entregar o seu software. Os containers são considerados o padrão para implantar microserviços e aplicações na nuvem (Ahmad Imtiaz, 2019).

Microserviços são itens intimamente relacionados com os containers. O mecanismo de container tem recebido muita atenção e é cada vez mais usado para implementar aplicações específicos. O mecanismo do container é inseguro devido à propriedade de compartilhamento do *kernel*, ele carece de uma avaliação concreta e sistemática de sua segurança usando e*xploits* reais.

As assinaturas de imagem de container são um recurso de segurança raramente implementado, embora o conteúdo das imagens esteja sempre mudando e seja difícil de entender, facilitando para os invasores ocultar conteúdo malicioso neles. A principal razão para isso é que o orquestrador de container mais popular Kubernetes não tem suporte nativo para assinaturas de imagem ou sua verificação (Belitz, 2020).

Para abordar o problema de segurança de containers e analisar os dados conhecidos disponíveis, vamos recorrer a alguns dos artigos publicados em conferências de TI e segurança da informação nos últimos 5 anos.

2.2 O stack de tecnologias dos containers.

Para entender a stack de implementação das tecnologias subjacente, usarei o diagrama da Figura 2, para o qual se descreve cada um dos seus elementos.

- **DevOps Tools**. São o conjunto de ferramentas que geralmente são executadas em uma máquina local para ajudar a configurar containers, máquinas virtuais ou empacotamento inicial de aplicações. Isso pode ser melhor chamado de "Ferramentas do desenvolvedor" ou "Ferramentas de operações" (REF). Um exemplo destas ferramentas é o Docker Machine. O Docker fornece uma ferramenta muito boa para facilitar a implantação e o gerenciamento de hosts Dockers em vários serviços de nuvem e hosts Linux chamados Docker Machine. O Docker Machine é instalado como parte do Docker Toolbox, mas pode ser instalado separadamente. (Smith, 2017).

- **Physical Host (or VM).** Esses são os computadores reais onde os containers serão executados. Pode ser um laptop local, um servidor de *data center* ou uma instância de nuvem pública. Vmware ESX, Microsoft Hyper-V, KVM, LXD (Smith, 2017).
- **Container OS**. Este é o sistema operativo (Linux ou Windows) que faz interface com o host físico subjacente e fornece os serviços no nível do SO que são compartilhados por todos os containers nesse host. CoreOS, Red Hat Atomic, RancherOS, Canonical Snappy, Vmware Photon (Andreas Aspernäs, 2016).
- **Container Runtime.** Esta é a camada de software que gerência containers (criar, iniciar, parar, destruir) em um host específico. Docker, Rkt, Lattice (René Peinl, 2016).
- **Containers.** Gere a interação entre a aplicação e o Container OS. Embora algumas pessoas façam uma analogia com uma máquina virtual, os containers não contêm todo o sistema operativo (Eder, 2016).
- **Marketplace / Image Management.** Esses são os servidores (executados no local ou como uma aplicação Cloud SaaS) que armazenam e gerenciam as imagens do container. Docker Registry, CoreOS Registry (Kelly Brady, 2020).
- **Configuration Management.** Estas são as ferramentas que gerem e automatizam os ficheiros de configuração e implantações. Ansible, Chef, Puppet, Docker Compose, Terraform (Michael Wurster, 2020).
- **Container Networking.** Também conhecido como "Software Defined Networking" (SDN), este é o framework que gerência o roteamento de pacotes entre containers. Docker, Networking, WeaveWorks, Flannel (CoreOS) (Miguel Borges de Freitas, 2020).
- **Container Cluster Management.** Esse conjunto de serviços ajuda a estabelecer clusters de containers que aparecem para o aplicativo como um único conjunto de serviços. Gere a disponibilidade dos recursos do container dentro de um cluster definido. Enxame Docker, Servo da Hashicorp (Bosco, 2020).
- **Service Discovery.** À medida que mais elementos/microserviços são adicionados a uma aplicação, pode ser difícil saber o que está disponível e onde está localizado. O Service Discovery gerencia o anúncio e as permissões de serviços à medida que chegam à rede. Hashicorp Consul, etcd (CoreOS). (Hausenblas, 2018).
- **Container Scheduling.** Este serviço procura o local mais eficiente para a colocação de containers em máquinas host. Gerência o planeamento de capacidade,

falha/reinicialização e dimensionamento de aplicações conforme requisitos ou mudanças de demanda. Kubernetes, Mesos. (Menouer, 2021).

- **Application Scheduling.** Trabalhando em estreita colaboração com o serviço Container Scheduling, este é um conjunto de serviços específicos de aplicações que descrevem os requisitos de como uma aplicação deve ser implantada (recursos, tempo, dependências). Marathon, Chronos, Hadoop, Spark (Sergey Belov, 2020).
- **Security.** Todos os elementos de segurança necessários nas arquiteturas de container do sistema operativo para serviços de autenticação para gerenciamento de dados. CoreOS Linux, Intel Clear Containers, Docker Notary, Hashicorp Vault (Syed, 2019).

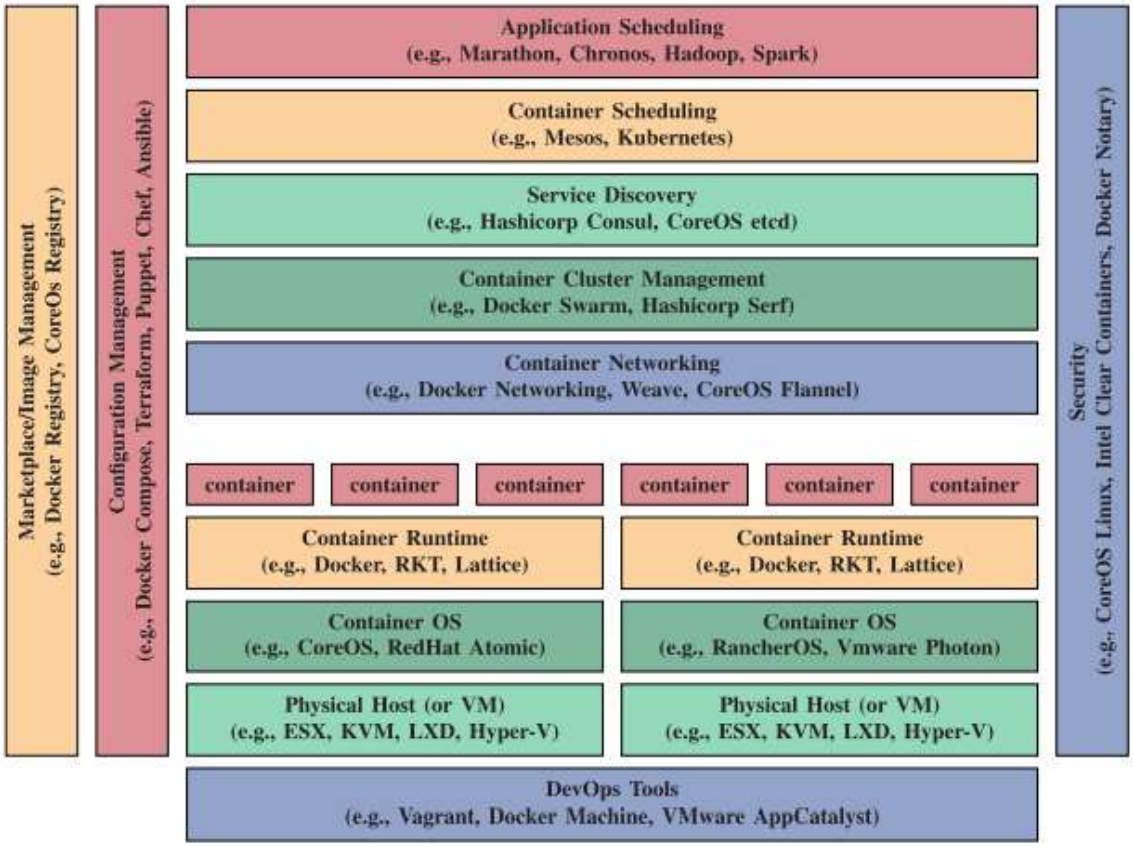

Figura 2. O Container stack e realização tecnologias

A figura 2 mostra que um container é um bloco de construção para uma stack de tecnologias maior que pode ser usada para facilitar as implantações de microserviços. (Ahmad Imtiaz, 2019). Olhando para o stack de baixo para cima, aqui está uma definição do que cada elemento contém, apresentando-se ainda alguns exemplos. (Miniman, 2015). Este

não inclui todas as realizações no mercado, mas destaca algumas das mais utilizadas atualmente.

2.3 Os cenários dos ataques e soluções.

Na maioria dos casos, os autores consultados destacam os tipos de ameaças à segurança no contexto de infraestruturas de containers que se sintetizam na Tabela 1 (Byungchul Tak, 2017).

Tabela 1. Prováveis vetores dos ataques e soluções.

Ameaça	**Possível ataque**	**Cenário possível**	**Solução possível**	**Referência(s)**
Imagens não confiáveis	Ataque tudo contido no host	Imagens não confiáveis representam uma séria ameaça aos containers no host. Um não confiável pode ter um scanner de vulnerabilidade pré-instalado que pode verificar todas as imagens na rede para encontrar vulnerabilidades e depois explorá-las	Use apenas imagens confiáveis usando assinaturas. Verificação de vulnerabilidades.	(Michael Falk, 2017)
Vulnerabilidades da aplicação	DoS em outros containers	Uma aplicação vulnerável pode causar ataques. DoS em containers vizinhos ou usar grande quantidade de recursos do host no qual afeta a operação de outros containers.	Verificação periódica de vulnerabilidades para imagens e aplicações	(Kumar, 2016)
Tráfego entre contiveres mal classificado	ARP-spoofing (MITM) e MAC flooding	Se os containers remotos de tráfego estiverem mal separados, isso pode permitir que um container comprometido execute MITM em outros containers	O container não deve ser capaz de se comunicar, a menos que seja necessário. A rede deve ser configurada com privilégios menos necessários.	(Panagiotis, 2020)
	DoS	Um container comprometido pode inundar a rede, o que torna outros containers inúteis ou	O container não deve ser capaz de se comunicar, a menos que seja necessário.	

		degradam muito o produtividade.		
Acesso ilimitado à rede de containers	Verificação de vulnerabilidade de porta	Se um container comprometido tiver acesso a uma rede de outros containers de vários níveis de sensibilidade, ele poderá executar facilmente a verificação de porta/vulnerabilidade e descobrir containers vulneráveis a serem direcionados.	Os containers devem ser separados em rede virtual com base no nível de sensibilidade. A monitorização da rede para anomalias (port scanning) também deve ser implementado.	(Zhecho D. Mitev, 2019)
Tempo de execução do container inseguro	Execução remota de código	Um invasor pode usar uma vulnerabilidade ou erro de configuração no tempo de execução para atingir outros containers	O tempo de execução do container deve ser monitorado quanto a vulnerabilidades e atualizado periodicamente	(Olufogorehan Tunde-Onadele, 2020)

2.4 Estudos publicados sobre esta temática

O Docker é popular na comunidade de desenvolvimento de software devido à versatilidade, portabilidade e escalabilidade dos containers. No entanto, as preocupações com as vulnerabilidades aumentaram à medida que a segurança das aplicações se tornou cada vez mais dependente da segurança das imagens que atendem aos blocos de construção das aplicações. À medida que mais processos de desenvolvimento migram para a nuvem, é fundamental validar a segurança das imagens extraídas de vários repositórios. Brady et. al, (2020) descrevem um sistema contínuo de alto produtividade e implantação contínua (CI/CD) que pressupõe que a segurança do Docker ocorre durante todo o ciclo de desenvolvimento de software. Os autores apresentam imagens com vulnerabilidades e medem a eficácia de sua abordagem para detetar e identificar vulnerabilidades e ameaças à segurança. Além disso, eles usam a análise dinâmica para avaliar a segurança dos containers do Docker de acordo com seu comportamento.

Sultan et. al (2019), detalham os princípios básicos da virtualização baseada em containers e fornecem vários exemplos de tecnologias de container. Em seguida, os autores fornecem uma análise detalhada da literatura sobre segurança e soluções de containers.

Como resultado do desenvolvimento e especificação de um modelo de ameaça, eles identificaram os vetores de ataque mais prováveis para containers e criaram quatro casos de uso genéricos que devem cobrir os requisitos de segurança em um ambiente de ameaças de container-host (Ahmad Imtiaz, 2019). Esta é uma investigação muito poderosa e profunda, que tomei como base teórica para minha dissertação.

Outro estudo importante para o meu trabalho foi publicado por Tunde-Onadele et. al, (2020) que fornece um estudo da eficácia de vários esquemas para detetar vulnerabilidades para containers. Especificamente, eles implementam e avaliam um conjunto de esquemas de deteção de vulnerabilidades estáticos e dinâmicos usando 28 explorações de vulnerabilidades do mundo real que são amplamente distribuídas em imagens Docker. Os seus resultados mostram que o esquema de varredura de vulnerabilidade estática deteta apenas 3 das 28 vulnerabilidades testadas, enquanto os esquemas de deteção de anomalia dinâmica detetam 22 explorações de vulnerabilidade. Assim, a combinação de esquemas estáticos e dinâmicos pode aumentar ainda mais a taxa de deteção para 86% (ou seja, 24 de 28 explorações).

O estudo de Sadique e Cassell (2021) é o que mais se aproxima do meu estudo, especialmente em relação ao problema declarado e ao método de solução semelhante que o pesquisador deveria resolver. E os ataques cibernéticos se tornaram um problema. Para se proteger contra eles, a troca de informações sobre segurança cibernética e a visualização de ameaças cibernéticas foram identificadas como importantes tópicos de pesquisa. Sadique e Cassell (2021) desenvolveram a plataforma CYBersecurity information Exchange with Privacy (CYBEX-P), que implementa desenvolvimentos em ambas as áreas como uma ferramenta de segurança colaborativa acessível. O gráfico de análise de ameaças CYBEX-P exibe indicadores de comprometimento e níveis de ameaça relatados por multidões. A interação intuitiva e eficiente com os dados é a chave para a adoção de tal sistema orientado a participantes e é o foco deste trabalho. Um estudo de usuário foi realizado com participantes de segurança cibernética para testar diferentes configurações de visualização. Medições relacionadas a variáveis dependentes, como precisão da tarefa e tempo de detecção de ameaças, foram registradas. Análises posteriores mostraram que o uso de cores localizadas para representar uma ameaça tem sérias limitações. Da mesma forma, a densidade da informação deve ser cuidadosamente considerada. Os investigadores concluíram que o uso indevido de propriedades visuais simples pode levar a uma diminuição

perigosa na precisão e no tempo de resposta, e fizeram recomendações sobre como evitar essas armadilhas (Farhan Sadique, 2021).

2.5. Ferramentas e tecnologias utilizadas

A falta de fontes confiáveis de informações detalhadas sobre as vulnerabilidades dos componentes de software de código aberto (OSS) é um grande obstáculo para manter uma cadeia de fornecimento de software segura e um processo eficaz de gerenciamento de vulnerabilidades. Fontes padrão de alertas e dados de vulnerabilidade, como o National Vulnerability Database (NVD), são conhecidas por sofrerem de baixa cobertura e qualidade inconsistente (Antonino Sabetta, 2018).

A segunda etapa tecnológica no desenvolvimento do protótipo backend foi a seleção de ferramentas de análise de vulnerabilidade estática para containers e aplicações desempacotadas. Para identificar problemas de segurança com o Docker existe um serviço de digitalização anteriormente conhecido como “ProjectNautilus”. O serviço fornece monitoramento automático, verificação e deteção de vulnerabilidades para imagens hospedadas no DockerHub. No entanto, o serviço não está disponível atualmente para todas as imagens do Docker (ou seja, não verifica imagens da comunidade). Portanto, é fundamental explorar as imagens do Docker que a comunidade criou. Além disso, várias ferramentas de varredura de containers estão disponíveis (como Clair, Anchore e Microscanner) que podem analisar imagens oficiais, bem como imagens da comunidade hospedadas no DockerHub. A abordagem adotada por essas ferramentas é que elas recolhem informações sobre pacotes (como pacotes básicos do OS3) e as comparam com um banco de dados de vulnerabilidades. Para demonstrar os problemas de vulnerabilidade tanto nas imagens oficiais quanto nas imagens da comunidade no DockerHub, foram analisados os pacotes do OS. No entanto, o estudo não examinou as vulnerabilidades encontradas na aplicação e as suas dependências empacotadas em um container. Como existem muitas ferramentas de varredura de container diferentes, é importante avaliar a qualidade dessas ferramentas para entender melhor seus pontos fortes e fracos. Tais avaliações são importantes para melhorar a abordagem utilizada pelas ferramentas de descoberta. É importante ter uma ferramenta de qualidade para resolver problemas de segurança, pois podemos estar a utilizar um componente no qual existe uma vulnerabilidade que pode ser corrigida em tempo útil (Omar Javed, 2021).

No entanto, para o frontend do projeto estava a procurar a ferramenta de nível mais baixo que idealmente estaria disponível como um comando Linux que seria passado por um túnel SSH. Portanto, voltei a minha atenção para a ferramenta Docker Bench for Security. É uma ferramenta para examinar o container do Docker em relação à especificação de segurança que baseia os seus testes no padrão da indústria (Jiang Wenhao, 2021).

NMAP scanner. O princípio da varredura de porta é determinar o status operativo da porta de destino retornando as características do pacote. De acordo com diferentes tipos de varredura, diferentes pacotes de sonda são gerados. O Nmap primeiro executa a operação de teste PING e envia os pacotes de teste necessários para a porta específica da máquina de destino. Em seguida, ele aguarda a necessidade de retransmitir o pacote de sondagem ou envia um novo pacote de sondagem. Finalmente, depende de diferentes tipos de métodos de deteção e espera receber diferentes tipos de pacotes de resposta (Si Liao, 2020).

Biblioteca Scapy. Scapy é um módulo Python que permite enviar, farejar e dissecar um pacote em uma rede. Esse recurso permite que os utilizadores criem uma aplicação que pode dissecar o funcionamento de um pacote de rede. Os pesquisadores criarão uma aplicação de aprendizagem de tráfego de protocolo em uma rede de computadores usando Scapy e linguagem natural para transmitir os resultados do processo de deteção em andamento. O aplicativo usa linguagem natural para transmitir a tradução do processo de *sniffing*. O resultado da tradução do processo de *sniffing* usando a linguagem natural de espera-se que esta aplicação seja eficaz e possa facilitar e fazer com que os utilizadores entendam e aprendam sobre o processo de trabalho do pacote de rede (Dharmesta, 2020).

Patches baseados em kernel. Patches baseados em kernel, como Anticap e Antidote, tentaram proteger contra falsificação de ARP no nível por host. O Anticap não permite atualizar o cache ARP do host com uma resposta ARP que contenha um endereço MAC diferente daquele que já está no cache. Infelizmente, isso também faz com que descarte respostas ARP gratuitas legais, o que é uma violação de ARP especificação de protocolo. Um antídoto para receber uma resposta ARP cujo endereço MAC o endereço for diferente do anteriormente armazenado em cache, ele tenta verificar se o MAC ainda está ativo. Se o endereço MAC obtido anteriormente ainda estiver ativo, então a atualização é rejeitada e o endereço MAC do invasor é adicionado à lista de banidos endereços.

Ambos os métodos referidos anteriormente dependem do fato de que a entrada ARP na cache é legal. Isso cria uma situação de corrida entre o hacker e a vítima. Se um invasor

obtiver sua entrada ARP falsa no cache dos hosts antes o host real, então o endereço MAC real é banido. Esta situação só pode ser desfeita por intervenção administrativa. Assim, podemos concluir que o treinamento incorreto pode fazer com que essas ferramentas falhem ao detetar falsificação de ARP.

Deteção passiva. Na descoberta passiva, detetamos as solicitações/respostas ARP na rede e criamos um banco de dados de mapeamento de endereços MAC/IP. Se notarmos uma mudança em qualquer um desses mapeamentos no tráfego ARP futuro, acionamos um alarme e concluímos que um ataque de falsificação de ARP está em andamento. A ferramenta mais popular nesta categoria - ARPWATCH. A principal desvantagem do método passivo é o intervalo de tempo entre a assimilação correspondência de endereços e deteção de ataques subsequentes. Em uma situação em que a falsificação de ARP começou antes que o descobridor fosse executado pela primeira vez, ou seja, a ferramenta examinará as respostas falsas em seu banco de dados de mapeamento IP/MAC. Agora, somente depois que a vítima começar a se comunicar com algum outro host, uma incompatibilidade será detetada e um alarme será acionado. Um atacante poderia ter escapado por causa desse atraso. Além disso, a entrada falsa sabia como no script acima ser cancelada manualmente pelo administrador da rede a única solução para esse problema é inserir manualmente os mapeamentos de endereço corretos ao banco de dados antes de executar a ferramenta ou criar tráfego de treinamento sem ataques. Ambos não são razoáveis devido a problemas de escalabilidade e portabilidade. Ideal, por exemplo, hosts móveis, por exemplo. laptops trazidos por clientes ou visitantes da empresa. Essa curva de aprendizado lenta torna impossível instalar ferramentas passivas em uma rede grande e esperar que detetem ataques instantaneamente. Os métodos passivos não têm inteligência e procuram cegamente uma incompatibilidade no tráfego ARP com suas tabelas de banco de dados aprendidas. Se a falsificação de ARP for detetada, não há como saber se a correspondência de endereço recentemente vista é o resultado de uma tentativa de falsificação ou se o endereço aprendido anteriormente era de fato falsificado. O método de aprendizado passivo também não é confiável. Novo mapeamento de endereço aprendido quando o tráfego ARP é visto a partir deles. Assim, a tabela de cache ARP do switch tentativa de estouro gerando pacotes de resposta ARP aleatórios por segundo com endereços MAC e IP arbitrários simplesmente levará à descoberta de novas estações em vez de atacar o tráfego. O método usa uma abordagem ativa para detectar falsificação de ARP. Enviamos ARP solicitação e pacotes TCP SYN para verificar. (Vivek Ramachandran, 2005).

bWAPP. O bWAPP é uma aplicação web com bugs, uma aplicação web de código aberto gratuito e intencionalmente inseguro. O termo bWAPP significa aplicativos da web com bugs. O bWAPP é de propriedade da IT SEC Games-Project0 e descreve uma aplicação da web deliberadamente com muitas falhas. O bWAPP foi desenvolvido por Malik Mesell com o objetivo de segurança de TI. Além disso, é de natureza lúdica e deve servir não apenas como treinamento, mas também como entretenimento. Esse aplicativo da Web intencionalmente desprotegido contém todas as vulnerabilidades de segurança conhecidas. Ele oferece a especialistas em segurança, desenvolvedores e estudantes a capacidade de detectar e prevenir problemas. Além disso, o bWAPP está se preparando para testes de penetração bem-sucedidos e projetos de hackers éticos. bWAPP é fácil de usar e muito fácil de entender. Existem 3 níveis de segurança de teste: fácil, médio e difícil. Existem mais de 100 vulnerabilidades da web disponíveis, portanto, todos os webs bugs conhecidos são cobertos. A ênfase não está em nenhum tópico específico.

Tabela 2. As características de bWAPP

Parâmetro	Valor do parâmetro
Bugs(Modelle)	over 100
Security levels 3	3
Challenges	91
Enviroment	English
Compatibility	Linux,
	Windows
	Mac
Fuzzing possibilities	Yes
Cheat Sheet	Available
Architecture	Open Source PHP application
	Backend MySQL database
	Hosted On Linux/Apache/IIS
	Supported on WAMP or XAMPP

O bWAPP é uma aplicação PHP de código aberto que usa um banco de dados MySQL. Pode ser hospedado em Linux ou Windows com Apache ou IIS e MySQL. Além disso, também pode ser instalado usando WAMP ou XAMPP. Outra opção é baixar o bee-box, uma máquina virtual com o aplicativo bW pré-instalado.

No bWAPP, a chamada é selecionada aqui primeiro. Em seguida, encontre um redireccionamento para a próxima página da Web para concluir a tarefa. As Figuras 8 e 9 mostram a interface de utilizadores do bWAPP. Na Figura 8 podemos ver uma lista suspensa

de bugs exploráveis para teste. E a figura 9 contém a página principal do bWAPP. (Kaimer, 2020)

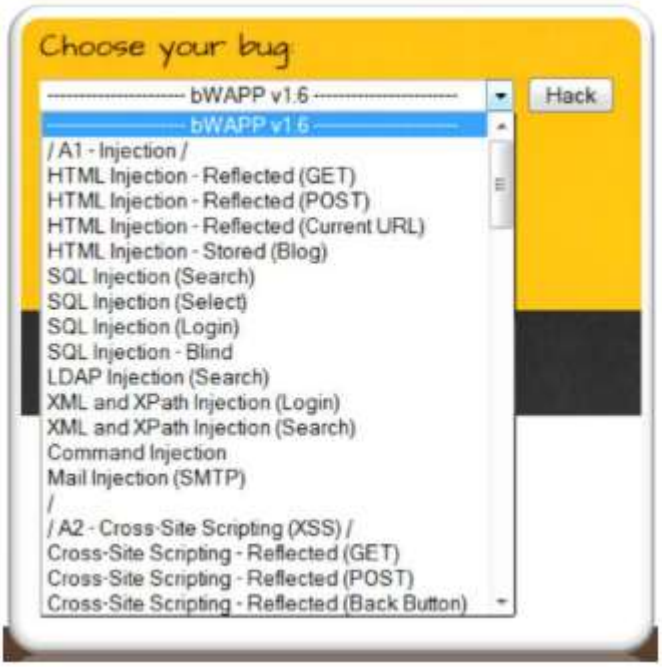

Figura 3. Lista de vulnerabilidades da bWAPP **(Kaimer, 2020)**

Figura 4.Página principal da bWAPP. **(Kaimer, 2020)**

Netmiko e Paramiko. Netmiko é essencialmente uma extensão do Paramiko. Em sua essência, a programação e automação de rede tem como principal objetivo simplificar as funções envolvidas na configuração, gerenciamento e operação de equipamentos de rede, topologias de rede, serviços de rede e conectividade de rede. Na minha configuração experimental posso usar o emulador GNS3 que é uma ferramenta para construir, e testar redes, capaz agora de projetar também conectar redes externas e permitir integração com imagens virtuais ou Docker Containers. Para minha implementação específica, use um

Photon OS que possui interpretador Python, permitindo-se configurar VM e automatizar a configuração e o monitoramento do ambiente do container por meio de ssh. O script Python é baseado nas bibliotecas Netmiko e Paramiko para controlar os containeres. Tanto o Netmiko quanto o Paramiko estão usando uma conexão SSH para obter o controle dos dispositivos. SSH (Secure Shell) Paramiko é uma implementação Python SSHv2, complementando a funcionalidade do protocolo cliente e servidor. É uma interface Python pura em torno dos conceitos de rede SSH e aproveita uma extensão Python C para criptografia de baixo. Netmiko é uma biblioteca de vários fornecedores, baseada no Paramiko, simplificando como um conjunto amplo de fornecedores e plataformas de rede.

Os scripts incorporam várias funcionalidades como criação de VLANs, protocolos de roteamento ou backup de configuração. Esses scripts podem ser usados para quase todos os dispositivos de rede, independentemente do fornecedor que os produz.

```
apt-get update
apt-get install python -y
apt-get install build-essential libssl-dev libffi-dev -y
apt-get install python-pip -y

pip install cryptography
pip install paramiko
pip install netmiko
```

Figura 5.Bibliotecas necessárias para o Photon OS Docker Container.

Como podemos ver, a diferença entre Netmiko e Paramiko é que Netmiko usa uma maneira mais fácil de se conectar usando ConnectHandler, que também usa SSH no backend. Além disso, quando usamos o Netmiko, temos que especificar o tipo de dispositivo que queremos controlar. (Paul MIHĂILĂ, 2017)

React é um framework JavaScript. O React foi originalmente criado por engenheiros do Facebook para resolver os desafios envolvidos no desenvolvimento de interfaces de utilizador complexas com conjuntos de dados que mudam ao longo do tempo. Este não é um empreendimento trivial e deve não apenas ser sustentável, mas também escalável para funcionar na escala do Facebook. O React nasceu na organização de anúncios do Facebook, onde eles estavam utilizando uma abordagem tradicional do Model-View-Controller do lado do cliente. Aplicativos como esses normalmente consistem em vinculação de dados bidirecional juntamente com o modelo de renderização. React mudou a maneira como essas aplicações foram criados fazendo alguns avanços ousados no desenvolvimento web.

O React não se propõe a resolver todos os problemas que encontrará no design da interface do utilizador e no desenvolvimento front-end. O React resolve um conjunto específico de problemas e, em geral, um único problema. Conforme declarado pelo Facebook e Instagram, o React constrói interfaces de utilizador em larga escala com dados que mudam ao longo do tempo.

O código é cada vez menos previsível e o tempo de desenvolvimento disparou. Este é exatamente o problema que o React se propõe resolver. No início, o React era um experimento mental. O Facebook pensou que já havia escrito o código de layout inicial para descrever como o aplicativo poderia e deveria ser, então por que não executar o código de inicialização novamente quando os dados ou o estado alterarem o aplicativo? provavelmente está se encolhendo agora porque sabe que isso significa que eles estariam sacrificando a produtividade e a experiência do utilizador. Ao substituir completamente o código em um navegador, verá cintilações na tela e flashes de conteúdo sem estilo. Apenas parecerá ineficiente. O Facebook sabia disso, mas também observou que o que ele criou – um mecanismo para substituir o estado quando os dados mudam – estava realmente funcionando até certo ponto. O Facebook decidiu então que, se o mecanismo de substituição pudesse ser otimizado, teria uma solução. Foi assim que o React nasceu como a solução para um conjunto específico de problemas (Gackenheimer, 2015).

Recharts. Recharts o é uma biblioteca de gráficos baseada em componentes construída com React e D3. D3, também conhecido como D3.js, é uma biblioteca JavaScript que permite editar vários documentos. Isso permite associar dados ao Document Object Model (DOM) e criar um documento de edição a partir dos dados. Por exemplo, D3 pode ser usado para criar uma tabela HTML de uma matriz JSON. também pode criar diagramas SVG. Ou seja, os gráficos para D3 foram importados para o React através do projeto Recharts. Recharts permite criar vários gráficos diferentes. A sua documentação contém um exemplo de cada diagrama, bem como o código-fonte para anexar o diagrama ao próprio projeto. As opções de edição de gráficos também são extensas. O utilizador pode personalizar a forma, tamanho da fonte e cor de vários elementos (Fig. 7). Os dados nos gráficos são recuperados de uma matriz contendo objetos (Ovaskainen, 2021).

Postman. O Postman é um aplicativo desenvolvido para testar diversas APIs, além de enviar diversas requisições ao servidor. Uma propriedade importante do programa é a capacidade de criar coleções de solicitações de API, o que acelera significativamente o desenvolvimento e otimiza o tempo e os recursos que muitas vezes são gastos na busca de

uma solicitação na memória ou no preenchimento manual de parâmetros. O Postman é fácil de trabalhar, e é por isso que muitos testadores o escolhem como a sua ferramenta de automatização. Para entender a estrutura de execução de consultas no Postman, tanto para execução padrão quanto para teste, vejamos os principais conceitos usados nele. Isso inclui Coleções, Pastas e Solicitações. Uma coleção é um arquivo de projeto que inclui todas as solicitações, possui seus próprios scripts e suas próprias variáveis. As coleções também mantêm um histórico de solicitações anteriores. A pasta combina solicitações em um grupo dentro da coleção, ela pode ter seus próprios scripts, mas não variáveis. As solicitações são criadas no construtor, mas também possuem seus próprios scripts. O esquema de interação desses elementos entre si é mostrado na Figura 9.

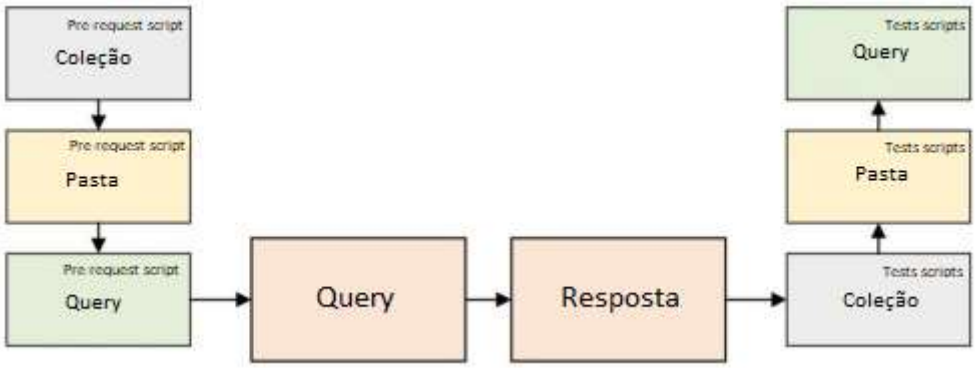

Figura 6.Diagrama de dependência no Postman

Além de executar solicitações, o Postman permite automatizar o teste de métodos de API, o que pode acelerar significativamente o processo.

desenvolvimento de produtos de software.

Para executar uma solicitação e um teste automatizado, deve:

1. Retire a URL e o token como variáveis de ambiente;

2. Anote o método necessário para a solicitação;

3. Especifique os parâmetros de solicitação;

4. Na guia Testes, escreva o código que realiza o teste

método (seus resultados).

5. Envie uma consulta;

6. Após concluir a solicitação, visualize os resultados: resposta (Corpo) e resultados do teste (Resultados do teste) Um dos testes mais simples e compreensíveis é verificar o status da solicitação concluída.

Se a solicitação for bem-sucedida, a resposta conterá um código de erro: 200.

Listagem de teste:

pm.test("Status code is 200", function () {

pm.response.to.have.status(200);

});

Após a conclusão bem-sucedida de tal teste, o utilizador pode ver o status do teste PASS, bem como a mensagem "Código de status é 200" (Sarycheva Yulia Yurievna, 2020).

Para testar a aplicação desenvolvida, também utilizei ferramentas especiais. Para desenvolver um teste completo, eu precisava criar um plano de teste com base nos requisitos e sprint backlog existentes. Já que o teste é uma espécie de testdrive de uma aplicação ou sistema recém-desenvolvido.

Plano de teste. Consiste numa documentação especificando o escopo, abordagem, recursos e cronograma das atividades de teste pretendidas.

Vale a pena distinguir entre depuração e teste. A execução de testes pode mostrar falhas causadas por defeitos no software. A depuração é uma atividade de desenvolvimento para localizar, analisar e corrigir esses defeitos. Testes confirmatórios subsequentes verificam se os defeitos corrigidos foram eliminados. Em alguns casos, os testadores são responsáveis apenas pelo teste inicial e pelo teste de confirmação final, enquanto os desenvolvedores realizam depuração e testes de componentes relacionados. No entanto, no desenvolvimento ágil e em alguns outros ciclos de vida, os testadores podem estar envolvidos na depuração e no teste de componentes. O padrão ISO (ISO/IEC/IEEE 29119-1) fornece informações adicionais sobre conceitos de teste de software (Olsen, Certified Tester Foundation Level Syllabus, 2018).

O teste não funcional de um sistema avalia características de sistemas e software, como usabilidade, eficiência de produtividade ou segurança. O padrão ISO (ISO/IEC 25010) permite obter uma classificação das características de qualidade do produto de software. Teste não funcional é o teste de “quão bem” o sistema se comporta. Ao contrário das perceções errôneas comuns, os testes não funcionais podem e geralmente devem ser

realizados em todos os níveis de teste e feitos o mais cedo possível. A descoberta tardia de defeitos não funcionais pode ser extremamente perigosa para o sucesso de um projeto. Técnicas de caixa preta podem ser usadas para derivar condições de teste e casos de teste para testes não funcionais. Por exemplo, a análise de valor limite pode ser usada para definir as condições de estresse para testes de produtividade. O rigor dos testes não funcionais pode ser medido através da cobertura não funcional. A cobertura não funcional é a extensão em que algum tipo de elemento não funcional foi exercido por testes e é expressa como uma percentagem do(s) tipo(s) de elemento coberto. Por exemplo, usando a rastreabilidade entre testes e dispositivos suportados para um aplicativo móvel, a percentagem de dispositivos que são abordados pelo teste de compatibilidade pode ser calculada, identificando potencialmente lacunas de cobertura. O design e a execução de testes não funcionais podem envolver habilidades ou conhecimentos especiais, como o conhecimento das fraquezas inerentes de um design ou tecnologia (por exemplo, vulnerabilidades de segurança associadas a linguagens de programação específicas) ou a base de utilizadores específica (por exemplo, as personas de utilizadores de sistemas de gestão de estabelecimentos de saúde) (Olsen, 2018).

A completude do teste funcional pode ser avaliada usando a cobertura da funcionalidade por testes. Cobertura de recurso é a extensão em que um tipo específico de item de recurso foi coberto por testes. O nível de cobertura é calculado como uma percentagem do número total de itens (ou tipos de itens). Por exemplo, usando a rastreabilidade dos testes e requisitos correspondentes, pode calcular a percentagem de requisitos cobertos pelos testes e identificar lacunas na cobertura (Olsen, 2018).

O teste de produtividade, que testa os atributos não funcionais do aplicativo sob teste (AUT), é uma atividade de teste importante. Da mesma forma, os termos "teste de produtividade" e "teste de requisitos não funcionais (NFR)" serão usados alternadamente para definir qualquer processo usado para testar, verificar ou relatar os atributos de qualidade (ou coloquialmente, as "-ilidades") da aplicação em teste (AUT) (Whiting, 2018).

Pytest.

```
        }
        actual = json
        assert expected == actual
```

No entanto, uma pergunta razoável pode surgir, por que devemos usar isso em vez do comando "*manage.py test"* do Django? A execução de uma suíte de testes com Pytest oferece alguns recursos não encontrados no mecanismo de teste padrão do Django:

- Menos clichê: não há necessidade de importar unittest, subclasse com métodos. Basta escrever testes como funções regulares.
- Gerencie dependências de teste com acessórios.
- Execute testes em vários processos para aumentar a velocidade.
- Existem muitos outros bons plugins para pytest.
- Troca fácil: os testes de estilo de teste de unidade existentes ainda funcionarão sem alterações.

Atlassian Jira. Um sistema popular de tratamento de casos é o Jira Software, uma ferramenta que suporta todo o processo de desenvolvimento de software, incluindo planeamento, desenvolvimento, rastreamento e relatórios. O sistema é construído sobre o conceito de problema, que representa um problema que precisa ser resolvido, após ser classificado de acordo com seu tipo (por exemplo, bug, melhoria ou história de utilizador). Normalmente, uma organização define os estados pelos quais um problema pode passar. Os problemas são compostos por um conjunto de campos de dados, incluindo prioridade (importância de um problema), resolução (a forma como um problema foi fechado) e relator (o funcionário que criou o problema). No Jira Software, o Scrum master pode criar novas pendências no backlog e associá-las a um sprint específico. Os desenvolvedores recebem problemas para trabalhar, de acordo com a distribuição em toda a equipe (Rita Marques, 2018).

TestLink é uma ferramenta que usada para gerenciar planos de teste, escrever casos de teste e relatar execuções de teste. Essa ferramenta atuou como editora e organizadora de casos de teste, armazenando todas as informações permitindo a atualização e criação de builds de teste com facilidade. Facilitou a documentação dos relatórios e também o controle das versões de execução dos testes (Eliane Figueiredo Collins, 2012).

Nmap, ou Network Mapper, é uma ferramenta gratuita e de código aberto disponível sob a GNU General Public License publicada pela Free Software Foundation. É mais comumente usada por administradores de rede e profissionais de segurança de TI para verificar redes corporativas em busca de hosts ativos, serviços específicos ou sistemas operacionais específicos. Parte da beleza do Nmap está em sua capacidade de criar pacotes

IP do zero e enviá-los usando metodologias exclusivas para executar os tipos de varredura mencionados acima e muito mais. Além disso, o Nmap vem com recursos de linha de comando ou GUI e é instalado facilmente em tudo, desde Unix e Windows até Mac OS X. Os requisitos de instalação dependem da versão do Nmap que está instalando e consistem principalmente em dependências de biblioteca de rede específicas para esta versão (Angela Orebaugh, 2018).

MÉTODO

Para registrar os resultados do estudo, bem como demonstrar claramente os objetivos e diretrizes deste trabalho, criarei um protótipo de sistema de monitorização de segurança de containers Docker. Este parágrafo descreve os métodos usados para desenvolver o protótipo.

Como base metodológica foi escolhido o Scrum. Os principais motivos da escolha Scrum foram:

- O pensamento enxuto no coração do Scrum reduz o desperdício e se concentra no essencial;
- O Scrum usa uma abordagem iterativa e incremental para otimizar a previsibilidade e controlar o risco.

Scrum é uma estrutura leve que ajuda pessoas, equipes e organizações a gerar valor por meio de soluções adaptáveis para problemas complexos. (Sliger, 2011). A Figura 4 mostra um esquema típico de gestão de projetos usando a metodologia Scrum. Após a adaptação ao projeto, foi utilizado este esquema para desenvolver um protótipo. O protótipo foi iniciado com o segmento de prototipagem onde foram definidos os artefactos do projeto.

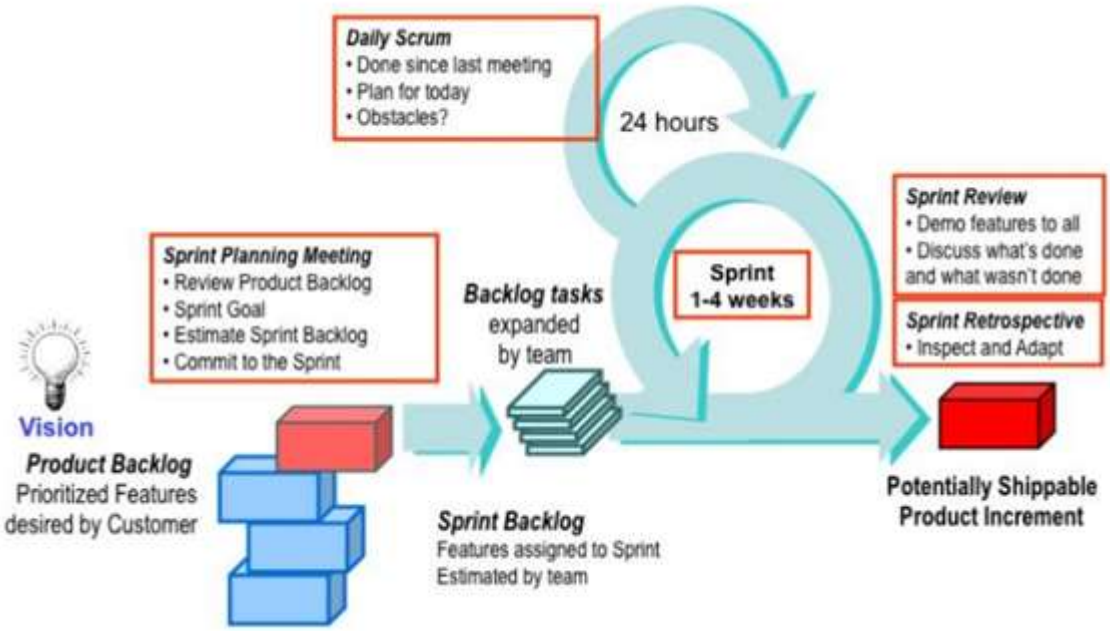

Figura 7. Esquema da gestão ágil de projetos com Scrum (Sliger, 2011)

Os artefactos do Scrum representam trabalho ou valor. Esses são projetados para fornecer a máxima transparência de informações importantes (figura 8) (Harris, 2021).

Figura 8. Artefactos Scrum (Harris, 2021)

3.1. Desenvolvimento de Product backlog e priorização de tarefas.

O backlog do protótipo contém uma lista dos seguintes itens:

- implantação e preparação de um stand com infraestrutura de container de teste Docker (VMWare Photon OS);
- definição de sensores e interfaces de software a partir dos quais os dados de segurança do Docker serão recolhidos;
- estudar a documentação do Python SDK for Docker e escrever código para extrair dados de segurança de uma lista de sensores aprovados;
- processamento e análise profunda dos dados coletados;
- programação de algoritmos para encontrar anomalias de segurança em containers;
- preparação do esqueleto do site Django;

- preparação do frontend React;
- montagem de todos os elementos na versão alfa;
- teste funcional e teste de produtividade do sistema e correção de bugs.

Na fase de formação do product backlog, as tarefas também foram priorizadas de acordo com o método MoSCoW. O próprio termo MoSCoW é um acrônimo derivado da primeira letra de cada uma das quatro categorias de priorização: M - *Must have* S - *Should have* C - C*ould have* W - *Won't have*. O método MoSCoW é uma técnica de priorização usada em gerenciamento, análise de negócios, gerenciamento de projetos e desenvolvimento de software para chegar a um entendimento comum com os stakeholders sobre a importância que eles atribuem à entrega de cada requisito; também é conhecido como priorização MoSCoW ou análise MoSCoW (Nicola Garzaniti, 2019).

Tabela 3.Priorização de tarefas do backlog do produto.

M Must have	▪ Desenvolvimento de modelo de ameaça ▪ Instalação e configuração baseado do Photon OS; ▪ Definir sensores e APIs para coletar uma coleção de dados de segurança ▪ Criação um ambiente de teste de microsserviço baseado no Docker; ▪ Desenvolvimento o code de master-container ▪ Montagem de todos os elementos na versão alfa; ▪ Teste funcional e teste de produtividade; ▪ Fixes de bugs.
S Should have	▪ Programação de algoritmos para encontrar anomalias de segurança em contauners; ▪ Desenvolvimento do frontend React;
C Could have	▪ Estudar a documentação do Python SDK for Docker; ▪ Processamento e análise profunda dos dados coletados
W Won’t have	▪ Suporte Alibaba, AWS, Azure

Depois de construir o backlog de produto protótipo e priorização de tarefas do backlog do produto, construí um sprint chart de Gantt. Cada sprint semanal contém a lista de trabalhos necessários para obter artefactos do Scrum (figura 9).

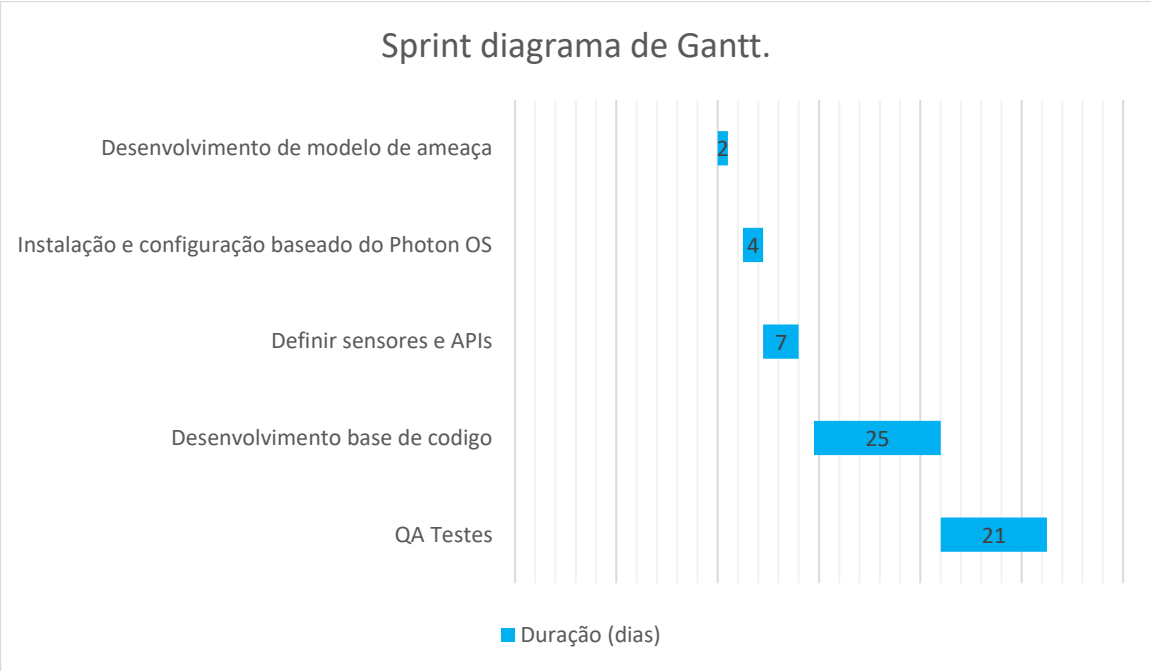

Figura 9. Sprint diagrama de Gantt.

Uma etapa importante do trabalho preliminar é a criação de um modelo de ameaça. A modelagem de ameaças visa identificar, comunicar e entender ameaças e eliminar bits

O modelo de ameaça é uma representação estruturada de todas as informações relevantes para a segurança que afetam a segurança. Em essência, este é um olhar para o aplicativo e seu ambiente através do prisma da segurança. A modelagem pode ser revelada para revelar uma variedade de coisas, incluindo a descoberta de aplicativos, sistemas, redes, sistemas distribuídos, dispositivos da Internet das Coisas (IoT) e processos de negócios.

O meu modelo de ameaça inclui:

- Descrição do objeto de simulação;
- Sugestões que podem ser testadas ou discutidas no futuro à medida que mudam;
- Sistemas de resposta potencial;
- Ações que podem ser necessárias para agir contra cada ataque;
- Como verificar o modelo e a ameaça e verificar o sucesso das ações tomadas.

A modelagem de ameaças é o processo de coletar, organizar e analisar todas essas informações. Aplicado ao software, esse permite que tome decisões informadas sobre os riscos de segurança do aplicativo. Além de criar um modelo, os esforços típicos de modelagem de ameaças também criam uma lista de melhorias de segurança prioritárias para o conceito, requisitos, design ou implementação de um aplicativo.

3.2 Objetivos da modelagem de ameaças

A modelagem de ameaças é uma família de atividades para melhorar a segurança, identificando ameaças e, em seguida, definindo contramedidas para prevenir ou mitigar os efeitos das ameaças em um sistema. Uma ameaça é um evento indesejado potencial ou real que pode ser malicioso (como um ataque dos) ou acidental (uma falha no dispositivo de armazenamento). A modelagem de ameaças é uma atividade planejada para identificar e avaliar ameaças e vulnerabilidades de aplicativos.

3.3 Modelagem de ameaças ao longo do ciclo de vida

A modelagem de ameaças é melhor aplicada continuamente ao longo de um projeto de desenvolvimento de software. O processo é quase o mesmo em diferentes níveis de abstração, embora as informações se tornem cada vez mais detalhadas ao longo do ciclo de vida. Idealmente, um modelo de ameaça de alto nível deve ser definido no início da fase de conceito ou planeamento e, em seguida, refinado ao longo do ciclo de vida. À medida que mais detalhes são adicionados ao sistema, novos vetores de ataque são criados e descobertos. O processo atual de modelagem de ameaças deve investigar, diagnosticar e remediar essas ameaças.

Essa é uma parte natural do aprimoramento do sistema para detetar novas ameaças. Ao escolher uma determinada tecnologia, como o Docker por exemplo, assumo a responsabilidade de identificar novas ameaças que essa escolha representa. Mesmo implementações, como o uso de expressões regulares para validação, podem criar novas ameaças potenciais que precisam ser abordadas.

A atualização de modelos de ameaças é útil após eventos como:

- Novo recurso lançado;
- Ocorre um incidente de segurança;
- Mudanças arquitetónicas ou infraestruturais.

3.4. Desenvolvimento de sprint backlog.

Os backlogs do sprint são criados selecionando uma tarefa do backlog do produto e dividindo essa tarefa em itens de sprint menores e acionáveis (Harris, 2021).

Dois métodos de revisão de product backlog são normalmente usados, revisão de especialistas ou criação de uma estrutura analítica do projeto. Essas estimativas destinam-se

especificamente a ser previsões e não medições exatas. Da mesma forma, o sprint backlog é o subconjunto de itens do product backlog que são definidos como parte do trabalho para um sprint específico. No entanto, ao contrário do backlog do projeto, o backlog do sprint é criado apenas pelos membros da equipe Scrum. Idealmente, o sprint backlog é atualizado todos os dias e não contém mais de 300 tarefas. A equipe pode precisar dividir uma tarefa se for determinado que levará mais de 16 horas. Além disso, a equipe pode determinar que os itens podem precisar ser adicionados ou subtraídos do sprint, mas isso é uma decisão da equipe, não é algo dirigido pelo proprietário do produto (Cervone, 2011).

Um esquema típico para transformar um backlog de produto em um backlog de sprint é mostrado na Figura 10.

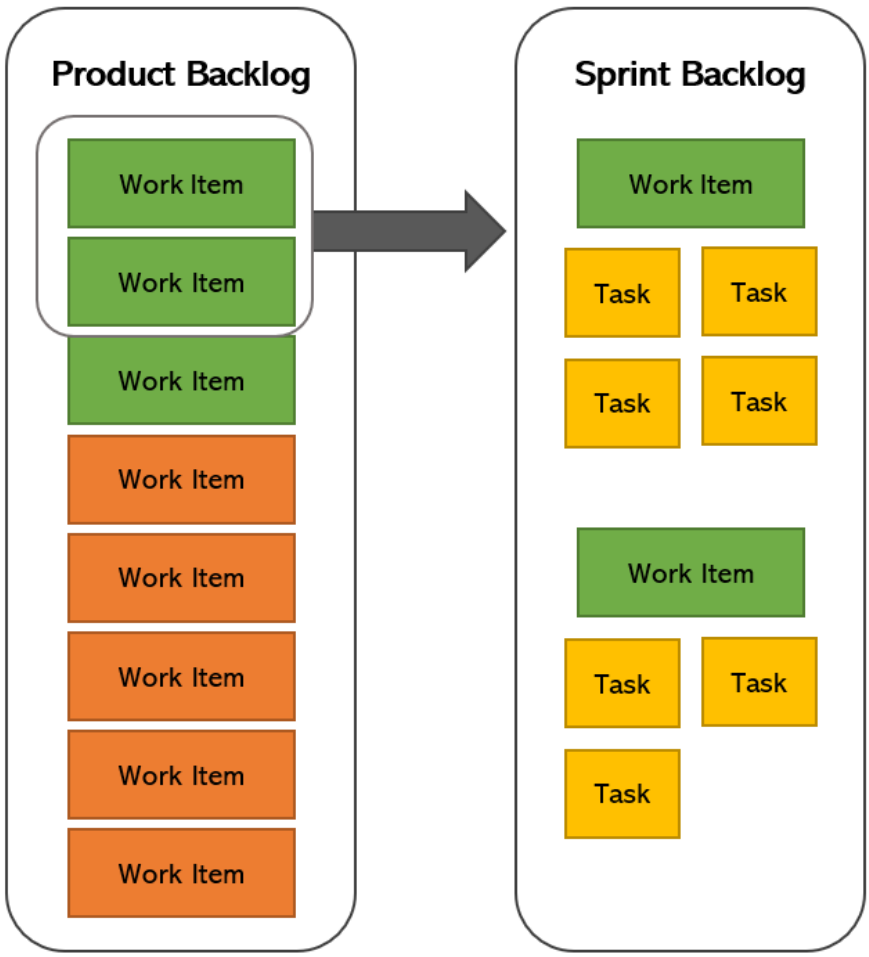

Figura 10. Transformação do Product backlog para o Sprint backlog

A Tabela 4 mostra a transformação do Product backlog para o Sprint backlog desenvolvido para cada sprint.

Tabela 4. Sprint backlog para desenvolvimento do protótipo.

Número do sprint	**Product backlog**	**Sprint backlog**		
1.	Desenvolvimento de modelo de ameaça	Desenvolvimento de um modelo básico de ameaças	Adaptação do modelo básico	
2.	Instalação e configuração baseado do Photon OS	Descarregar uma imagem Photon OS de distribuição com versão atual.	Implantação da máquina virtual de VmWare Photon OS	Verificação de configuração. Instalando os Linux daemons necessários (Docker, Python).
3.	Criação um ambiente de teste de microsserviço baseado no Docker	Definir os critérios para serviços de teste, com diferentes níveis de segurança inicial.	Selecionando aplicações de container de teste e implantando serviços do docker hub.	Criar uma configuração vulnerável obviamente de um ambiente em container para demonstrar visível como a monitorização funciona
4.	Definir as interfaces para coletar uma coleção de dados de segurança	Definir as interfaces para obter parâmetros dos containers e imagens.	Definir as interfaces para obter parâmetros do OS básica.	Definir as interfaces para obter parâmetros de rede
5.	Programação de algoritmos para encontrar anomalias de segurança em containers.	Criar uns algoritmos.	Desenvolvimento o code de backend	
6.	Montagem de todos os elementos na versão alfa.	Desenvolvimento o code de frontend	Desenvolvimento o máster-container e colocar isso no Docker Hub.	Criar documentação para utilizadores
7.	Teste funcional e teste de produtividade	Implantação rápida de master-container do Docker Hub.	Realizar pentestes.	
8.	Fixes de bugs	Correção de bugs na parte funcional e produtividade	Correções de bugs na parte UX	

3.5. Preparação do ambiente para o desenvolvimento do protótipo

Durante o desenvolvimento do protótipo, foi necessário desenvolver uma base de código para um protótipo de sistema baseado na web para monitorização dos parâmetros de segurança de um ambiente de container. Para iniciar o desenvolvimento, precisava de

requisitos uma infraestrutura de teste e dados de origem para codificação. Para aumentar a confiança e a conscientização sobre os métodos e técnicas utilizados na identificação de sinais de ataques cibernéticos, segue abaixo um mapa das tecnologias em forma de relatório para cada item listado na Tabela 1 e na Tabela 2 e mesmo como de acordo com o plano de Scrum de desenvolvimento apresentado na Tabela 3. O relatório também fornece uma justificativa tecnológica para a escolha de um ou outro método para detetar sinais de deterioração na segurança da infraestrutura e melhorar a qualidade dos dados iniciais. Usando os exemplos do relatório, pode ser verificado a cada vetor de ataque, verificar como funciona separadamente e como parte de um conjunto de detetores.

3.5.1. Instalação e configuração baseado do Photon OS.

O Photon OS é um sistema operativo Linux de código aberto minimalista da VMware otimizado para plataformas de computação em nuvem, implantações VMware vSphere e aplicativos nativos da nuvem. O Photon OS é um container de host Linux otimizado para vSphere e plataformas de computação em nuvem, como Amazon Elastic Compute e Google Compute Engine. Como um sistema operativo leve e extensível, o Photon OS funciona com os formatos de container mais comuns, incluindo Docker, Rocket e Garden. O Photon OS inclui um sistema de gerenciamento de ciclo de vida de pacotes compatível com yum chamado tdnf. Quando usado com ferramentas e ambientes de desenvolvimento, como VMware Fusion, VMware Workstation e runtimes de produção (vSphere, vCloud Air), o Photon OS permite que mova aplicativos baseados em contêiner do desenvolvimento para a produção. Com seu tamanho pequeno e tempos de inicialização e execução rápidos, o Photon OS é otimizado para computação em nuvem e aplicativos em nuvem. O Photon OS está disponível nos formatos binários pré-empacotados apresentados na Tabela 5.

Tabela 5. Formatos binários pré-empacotados do Photon OS

Formato	Descrição
ISO Imagem	Imagem ISO Contém tudo o que precisa para instalar uma instalação mínima ou completa do Photon OS. A imagem ISO inicializável tem uma instalação manual ou pode ser usada com ambientes PXE/kickstart para instalação automática.

OVA	OVA Ambiente mínimo pré-instalado configurado para ambientes de hypervisor VMware. Esses ajustes incluem um kernel totalmente limpo e otimizado para melhorar a produtividade de inicialização e execução de contêineres e aplicativos Linux. Como o OVA é a definição completa de uma máquina virtual, disponibilizamos o Photon OS OVA com hardware virtual versão 11; isso garantirá a compatibilidade com várias versões de plataformas VMware ou permitirá que aproveite os melhores e mais recentes aprimoramentos de hardware virtual.
Amazon AMI	Amazon AMI Uma versão pré-empacotada e testada do Photon OS, pronta para ser implantada em seu ambiente de nuvem Amazon EC2. Costumávamos postar documentação sobre como criar uma instância compatível com a Amazon, mas agora fizemos o trabalho para s.
Google GCE Image	Imagem do Google GCE Imagem do Google GCE pré-empacotada e testada, pronta para ser implantada em seu ambiente do Google Compute Engine, com todas as alterações e requisitos de pacote para executar o Photon OS no GCE.
Azure VHD	VHD do Azure Uma imagem HD do Azure pré-empacotada e testada, pronta para ser implantada na nuvem do Microsoft Azure, com todas as alterações e requisitos de pacote para executar o Photon OS no Azure.

Como o ambiente de teste do desenvolvimento é baseado no VMware ESXi, o modo de implantação do Workstation pode ser usar o Photon OS como uma máquina virtual no VMware Workstation. Foi descarregado o Photon OS como um ficheiro OVA ou ISO e instalada a distribuição do Photon OS no vSphere. Depois do Photon OS instalado, foi implantada uma aplicação em container no Docker com um único comando.

Depois de criar e instalar a imagem ISO para o Photon OS e particionar o disco, continuei com a instalação. O instalador me pediu para selecionar uma opção de instalação, conforme se mostra na Figura 11.

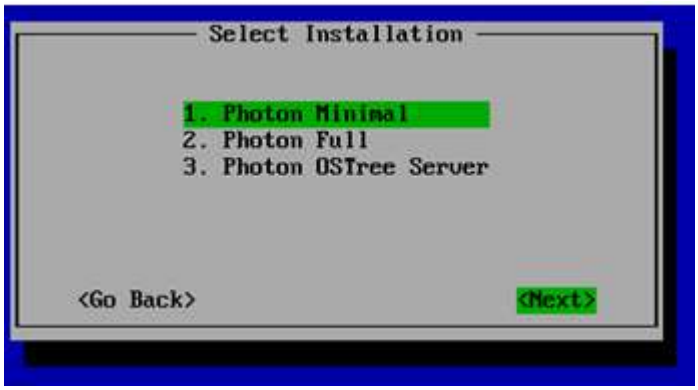

Figura 11. Opções de instalação de Photon OS.

3.5.2. Criação um ambiente de teste de microsserviço baseado no Docker.

Para um processo de desenvolvimento completo, foi necessário criar um ambiente de teste com containers em execução constante. Depois de fazer um pequeno trabalho preparatório, foi escolhido o bWAPP (http://www.itsecgames.com/) na forma de um container Docker.

No Photon OS, tdnf é o gestor de pacotes padrão para instalar novos pacotes. Esta é uma implementação C do gestor de pacotes DNF sem dependências do Python. DNF é a próxima versão principal do yum. O Tdnf está presente nas versões mínima e completa do Photon OS. Tdnf lê repositórios yum e funciona como yum. A versão completa do Photon OS também inclui o yum, e pode instalar pacotes com o yum. Na versão mínima do Photon OS, podemos gerenciar pacotes com o yum, mas primeiro foi necessário instalá-lo executando o seguinte comando tdnf como root:

```
tdnf install yum
```

Tdnf realiza um subconjunto dos comandos dnf listados no manual dnf (Figura 12).

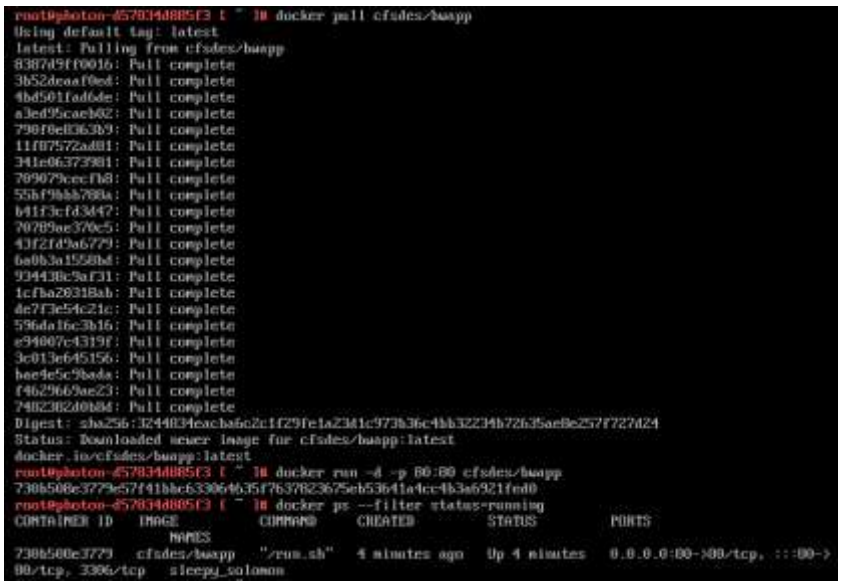

Figura 12. Implementação e start do container com bWAPP.

3.6. Teste funcional e teste de produtividade.

Os principais objetivos de testar o protótipo, assim como qualquer projeto, podem incluir:

- Avaliação de produtos de trabalho, como requisitos, histórias de utilizadores, projeto e código.
- Verificando se todos os requisitos especificados foram atendidos.
- Verificar se o item de teste está completo e funciona conforme o esperado pelos utilizadores e pessoas interessadas.
- Construir confiança no nível de qualidade do item de teste.
- Prevenção de defeitos.
- Deteção de falhas e defeitos.
- Fornecer às partes interessadas informações suficientes para permitir que tomar decisões informadas, especialmente em relação ao nível de qualidade da instalação testando.
- Reduzindo o risco de qualidade de software inadequada (por exemplo, falhas de trabalho perdidas).

Procedimentos de teste. Documentação detalhando procedimentos de teste, entradas, resultados previstos e condições de execução para cada item de teste.

Validação. Processo de avaliação de um sistema no final do processo de desenvolvimento para garantir a conformidade com os requisitos funcionais e de

produtividade. Baseia-se na avaliação de um teste integrado de hardware e software. O processo de validação garante que os recursos do sistema, conforme descrito no documento de especificação de requisitos, sejam implementados no hardware e no software.

Verificação. Processo para determinar se o produto de uma determinada fase do ciclo de desenvolvimento de um sistema atende aos requisitos estabelecidos para aquela fase. A verificação é baseada na documentação de desenvolvimento do sistema. A verificação garante que uma tradução precisa das informações de uma fase de desenvolvimento para a próxima tenha sido realizada (por exemplo, a revisão das especificações do projeto garante que o projeto atenda aos requisitos funcionais).

Para realizar os testes, utilizaram-se ferramentas conhecidas como: Jira, Pytest, Testlink, Kali Linux e as suas ferramentas, tais como: Nmap, hping3,

O Pytest é uma ferramenta de teste Python robusta, o Pytest pode ser usado para todos os tipos e níveis de teste de software. O Pytest pode ser usado por equipes de desenvolvimento, equipes de controle de qualidade, grupos de testes independentes, indivíduos praticando TDD e projetos de código aberto. Projetos em toda a Internet mudaram de unittest ou nose para Pytest, incluindo Mozilla e Dropbox. Porque o Pytest oferece recursos poderosos, como reescrita 'assert', um modelo de plug-in de terceiros e um modelo de fixture poderoso e simples que é incomparável em qualquer outra estrutura de teste. Pytest é uma estrutura de teste de software, o que significa que Pytest é uma ferramenta de linha de comando que encontra automaticamente os testes que escrevemos, executa os testes e relata os resultados. tem uma biblioteca de guloseimas que pode utilizar em seus testes para ajudá-lo a testar com mais eficiência. O Pytest pode ser estendido escrevendo plugins ou instalando plugins de terceiros e pode ser utilizando para testar distribuições Python. E integra-se facilmente com outras ferramentas como integração contínua e automação web. Aqui estão algumas das razões pelas quais o Pytest se destaca acima de muitos outros frameworks de teste:

- Testes simples são simples de escrever em Pytest;
- Testes complexos ainda são simples de escrever;
- Os testes são fáceis de ler;
- Os testes são fáceis de ler. (Tão importante que é listado duas vezes.);
- Pode começar em segundos;

- Pode utilizar assert para falhar em um teste, não coisas como self.assertEqual() ou self.assertLessThan();
- Apenas afirmar.

Pode utilizar Pytest para executar testes escritos para unittest ou nose. O Pytest está sendo desenvolvido e mantido ativamente por uma comunidade apaixonada e crescente. É tão extensível e flexível que caberá facilmente em seu fluxo de trabalho. E porque está instalado separadamente da sua versão do Python, pode usar a mesma versão mais recente do pytest no legado Python 2 (2.6 e superior) e Python 3 (3.3 e superior). (Okken, 2017).

Os ataques de flood SYN podem ser criados utilizando a ferramenta de software hping3 pura. O mesmo design pode utilizar para criar muitos tipos diferentes de ataques, como spoofing, no spoofing, camada 3, ataques de camada 4, como ataque de flood TCP, ataque UDP, ataque de flood ICMP, ataque TCP SYN-ACK, TCP FIN-ACK ataque, etc. Tais ataques são necessários para avaliar de produtividades de sistemas de segurança. (Shaila R Ghanti, 2015).

No teste funcional basicamente é feito o teste das funções do componente ou sistema. Refere-se a atividades que verificam uma ação ou função específica do código. O teste funcional tende a responder a perguntas como “o utilizador pode fazer isso” ou “esse recurso específico funciona”. Isso geralmente é descrito em uma especificação de requisitos ou em uma especificação funcional. As técnicas usadas para testes funcionais geralmente são baseadas em especificações. A funcionalidade de teste pode ser feita de duas perspetivas:

- Teste baseado em requisitos. Neste tipo de teste os requisitos são priorizados dependendo dos critérios de risco e, consequentemente, os testes são priorizados. Isso garantirá que os testes mais importantes e críticos sejam incluídos no esforço de teste.
- Testes baseados em processos de negócios. Neste tipo de teste são descritos os cenários envolvidos na utilização comercial diária do sistema. Usa o conhecimento dos processos de negócios.

O teste funcional do sistema inclui testes para avaliar as funções que devem ser executar sistema. Requisitos funcionais podem ser descritos em produtos de trabalho (requisitos, especificação, necessidade de negócios, história do utilizador, cenário uso) ou na especificação funcional, ou pode não estar documentado. As funções do sistema fornecem uma resposta à pergunta "o que o sistema faz".

Os testes funcionais devem ser executados em todos os níveis de teste (por exemplo, testes para componentes do sistema podem ser baseados em uma especificação de componente), portanto o foco do teste é diferente para cada nível. O teste funcional considera o comportamento do sistema, portanto, técnicas de teste de caixa preta podem ser usadas para obter condições de teste e casos de teste.

A eficiência da produtividade (ou simplesmente "produtividade") é parte integrante do fornecimento de uma "boa experiência" para os utilizadores quando eles usam seus aplicativos em várias plataformas fixas e móveis. O teste de produtividade desempenha um papel crítico no estabelecimento de níveis de qualidade aceitáveis para o utilizador final e geralmente está intimamente integrado a outras disciplinas, como engenharia de usabilidade e engenharia de produtividade. Além disso, avaliar a operatividade, usabilidade e outras características de produtividade sob condições de carga, como durante um teste de produtividade, pode revelar problemas específicos de carga que afetam essas características. O teste de produtividade não se limita ao domínio da Web em que o foco está no utilizador final. Também é relevante para diversas áreas de aplicação com diferentes arquiteturas de sistema como o clássico cliente-servidor, distribuído e embarcado. Tecnicamente, a eficiência de produtividade é classificada no modelo de qualidade de produto ISO 25010 [ISO25000] como uma característica de qualidade não funcional com três subcaracterísticas, tais como:

- Comportamento ao longo do tempo. Como regra geral, é mais comum avaliar o comportamento do tempo. Este aspeto de testar o desempenho de dureza dos componentes ou a resposta de um sistema às entradas do utilizador ou do sistema em um determinado tempo e sob condições especificadas. O tempo de execução pode ser medido desde o tempo de ponta a ponta que o sistema leva para calcular a entrada do utilizador, até o número de ciclos de CPU, um componente de software padrão para executar uma tarefa específica.
- Utilização de recursos. Se o acesso aos recursos do sistema for identificado como um risco, o uso desses recursos (por exemplo, uso de RAM limitada) pode ser investigado executando testes de desempenho específicos.
- Capacidade. Se problemas de comportamento do sistema nos limites de largura de banda do sistema exigidos puderem ser (por exemplo, número de utilizadores ou volumes de dados) importantes como risco, testes de desempenho foram realizados para avaliar a avaliação da arquitetura do sistema.

Vários tipos de testes de desempenho podem ser definidos. Cada um deles pode ser aplicável a um determinado projeto, dependendo da finalidade do teste.

- Teste de desempenho é um termo geral que inclui qualquer tipo de teste focado no desempenho (responsividade) de um sistema ou componente sob diferentes quantidades de carga.
- Teste de carga concentra-se na capacidade do sistema de lidar com níveis crescentes de cargas realistas esperadas como resultado de solicitações de transação geradas por um número controlado de utilizadores ou processos simultâneos.
- Teste de estresse se concentra na capacidade de um sistema ou componente de lidar com cargas de pico que estão dentro ou além das cargas de trabalho esperadas ou especificadas. O teste de estresse também é usado para avaliar a capacidade do sistema de lidar com disponibilidade reduzida de recursos, como poder de processamento disponível, largura de banda disponível e memória.
- Teste de escalabilidade concentra-se na capacidade do sistema de atender a requisitos de desempenho futuros que podem ser maiores do que os exigidos atualmente. O objetivo desses testes é determinar a capacidade do sistema de crescer (por exemplo, com mais utilizadores, mais dados armazenados) sem violar os requisitos de desempenho atualmente estabelecidos ou travar. Uma vez que os limites de escalabilidade são conhecidos, os limites podem ser definidos e monitorados em produção para fornecer avisos de problemas que podem estar prestes a ocorrer. Além disso, o ambiente de produção pode ser ajustado com a quantidade adequada de equipamentos.
- Teste de pico concentra-se na capacidade do sistema de responder adequadamente a rajadas repentinas de cargas de pico e, posteriormente, retornar a um estado estável.
- Teste de resistência concentra-se na estabilidade de um sistema durante um período de tempo específico para o contexto operativo do sistema. Esse tipo de teste verifica se não há problemas de capacidade de recursos (por exemplo, vazamentos de memória, conexões de banco de dados, pools de threads) que possam prejudicar o desempenho e/ou causar falhas nos pontos de interrupção.
- Teste de simultaneidade se concentra no impacto de situações em que certas ações ocorrem ao mesmo tempo (por exemplo, quando um grande número de utilizadores faz logon no sistema ao mesmo tempo). Problemas de simultaneidade são notoriamente difíceis de encontrar e reproduzir, especialmente quando o problema

ocorre em um ambiente onde o teste tem pouco ou nenhum controle, como um ambiente de produção.

- Teste de capacidade determina quantos utilizadores e/ou transações um determinado sistema suportará e ainda atenderá às metas de desempenho estabelecidas.

No Agile e em outros ciclos de vida iterativo-incrementais, a equipe deve identificar testes de desempenho estáticos e dinâmicos em iterações iniciais, em vez de esperar que as iterações financeiras consumam o risco de produtividade.

3.7. Fixes de bugs.

Os dois principais parâmetros que formam a base para o rastreamento e resolução eficazes de defeitos são:

- Prioridade de defeitos;
- Gravidade de defeitos.

Esses termos são quase intercambiáveis não apenas para teste, mas também para desenvolvimento. No contexto de defeitos, a prioridade de um defeito indica a urgência com que ele precisa ser corrigido. A gravidade, por definição, é usada para descrever a gravidade de um evento indesejado. Portanto, quando se trata de bugs, a gravidade do bug indicará seu impacto no sistema em termos de impacto. Prioridade e gravidade têm algumas classificações entre elas que ajudam a determinar como um defeito deve ser tratado. Muitas organizações usam diferentes ferramentas de relatório de defeitos para que os níveis possam variar, utilizei o Atlassian Jira para isso e o esquema de classificação default de 4 níveis para prioridade e gravidade.

DESENVOLVIMENTO DO PROTÓTIPO

4.1 Definição das interfaces para obtenção uma coleção de dados de segurança.

Existem várias ferramentas que podem receber tráfego e gerar um log de overhead extraindo as informações relevantes dos pacotes. Por exemplo, o conteúdo de uma entrada CLF é passado entre um cliente HTTP e um servidor HTTP.

As ferramentas de análise de rede fornecem ferramentas de divisão de pacotes ou recuperação de sessão para análise profunda de pacotes. Eles criam um modelo de sessão com base nos dados do pacote. Essas ferramentas são muito úteis para ter uma ideia aproximada do que acontece durante uma sessão na ausência de um log de serviço, mas são bastante padrão quando se trata de restaurar uma sessão de rede: não funcionam com dados criptografados, apenas fornecem dados de sessão aproximados e podem ignorar detalhes de implementação, enquanto a restauração do processo é bastante cara.

Ao mesmo tempo, esses programas de recolha de dados funcionam com quaisquer dados de tráfego de rede e não requerem o processo logisticamente complexo de definir e instalar um serviço separado. Todos os sensores de rede NetFlow, tcpdump e IDS funcionam no nível da rede, mas a saída é diferente para todos. Para entender a diferença entre essas camadas, considere as interações do protocolo HTTP em termos de sensores de três níveis diferentes: um sensor que analisa os pacotes e é instalado no nível da rede; um sensor em nível de host que monitoriza a produtividade e controla o acesso aos arquivos; e, finalmente, o log do servidor HTTP.

O sensor de rede vê os pacotes que foram enviados, mas não os agrupa em estruturas de protocolo HTTP, como sessões, cookies ou páginas. O sensor do host pode determinar quando um arquivo foi acedido pela última vez, mas não associa esse arquivo a uma URL ou solicitação. O sensor de serviço pode mostrar a existência de uma sessão HTTP e a página concluída, mas não detecta uma varredura de porta 80 pendente), o restante só pode fornecer informações ao especialista em segurança para hipóteses. Outras coisas sendo iguais, é melhor ter um sensor localizado o mais próximo possível do alvo. Os níveis dos sensores e seus campos de visão determinam a redundância na operação das combinações de sensores. Se dois sensores estiverem localizados no mesmo nível e a área. Se o campo de visão de um for maior que o do outro, então um sensor com campo de visão menor é redundante e talvez

não deva ser usado. Por outro lado, se dois sensores têm o mesmo campo de visão, mas estão localizados em níveis diferentes, eles devem se complementar. O sensor pode coletar dados em um dos três níveis:

- **Rede.** Sensores deste nível recolhem informações sobre o tráfego de rede. Exemplos de tais sensores incluem VPNs, a maioria dos sistemas de deteção de intrusão (IDSes), programas de coleta de dados do protocolo NetFlow, como YAF, bem como programas de coleta de dados TCP, como Snort e dados brutos tcpdump.
- **Host.** Sensores de host monitoram processos em andamento no host, como login, logout, acesso a arquivos etc. Um sensor de host pode fornecer informações que um sensor de rede não fornece, como acesso real a um host específico ou sobre o uso de periféricos USB externos. Os sensores de host incluem ferramentas de sistema de prevenção de intrusão (IPS), como Tripwire ou o aplicativo HIPS da McAfee, bem como logs de sistema e de segurança. Os sensores de host fornecem informações sobre operações de baixo nível, mas não revelam muito sobre a execução de serviços. Obviamente, só podemos usar esses sensores em hosts que se conhecem. Os hosts não autorizados devem ser detetados antes que possam ser verificados.
- **Serviço.** Os sensores de serviço são gerados por determinados processos, como logs de servidor HTTP e SMTP. Os sensores de serviço monitoram eventos bem formados, se não legítimos, dentro do serviço (por exemplo, um sensor HTTP capturará uma tentativa de URL com falha, mas não registrará uma sessão da porta 80 que não enviou comandos compatíveis com HTTP. sensor relacionado a sensores comuns, sensores de serviço capturam mais interações com um serviço específico: enviar e-mails, fazer requisições HTTP, etc.

A abordagem descrita acima é típica para hosts clássicos e também em maior medida para hosts de virtualização de hipervisor. Para a virtualização de containers, como podemos ver, é necessário esclarecer o funcionamento dos sensores no nível do host. Como, até onde sei, não há uma descrição geralmente aceita do modelo de segurança com containers, tentei separar o subnível de container no nível de host. Devido ao conjunto limitado de dependências e isolamento básico contido em cada tecnologia de container, decidi focar o campo de visão do sensor diretamente no container, alvo dos invasores.

Muitas organizações diferentes tentaram desenvolver o Common Vulnerability Detection Standard (CIDF) e o Vulnerability Detection Message Protocol (IDMEF). Mas nenhum deles atingiu uma implementação industrial séria. O que tem sido amplamente

aceito é a CEF. Originalmente desenvolvido pela Arcsight (agora parte da Hewlett-Packard), o sensor é um formato padrão para enviar mensagens para seu SIEM. CEF é um formato de registro que define eventos de segurança usando um cabeçalho numérico e uma série de pares chave/valor. Por exemplo, uma mensagem CEF para um ataque do host 192.168.1.1 pode ter esta aparência:

CEF:0|OMeu detector de ataque|Teste|1.0|1000|Ataque|5|src=192.168.1.1

O CEF é independente de transporte, mas a maioria das implementações do CEF usa syslog como transporte preferencial.

Por isso é o primeiro requisito para realizar em protótipo. Para realizar um detetor de assinatura, é necessário analisar informações de baixo nível sobre repositórios assinados. Isso inclui todas as tags de imagem assinadas, quem as assinou e quem pode assinar novas tags. A Figura 13 mostra o exemplo de código para obter de assinatura de imagem MySQL em formato JSON.

```
root@photon-d57834d885f3 [ ~ ]# docker trust inspect mysql:latest
[
    {
        "Name": "mysql:latest",
        "SignedTags": [
            {
                "SignedTag": "latest",
                "Digest": "1c75ba7716c6f73fc106dacedfdcf13f934ea8c161c8b3b3e4618bcd5fbcf195",
                "Signers": [
                    "Repo Admin"
                ]
            }
        ],
        "Signers": [],
        "AdministrativeKeys": [
            {
                "Name": "Root",
                "Keys": [
                    {
                        "ID": "531d6fc3ea002c6962de35933d0067f193d61955147fe21829e12c8fc96e5c5e"
                    }
                ]
            },
            {
                "Name": "Repository",
                "Keys": [
                    {
                        "ID": "7363fd30caf3d5c407d8e779fcfa7e5c0987335e391591e4aa4a04c6abdb4895"
                    }
                ]
            }
        ]
    }
]
```

Figura 13.Obtenção de dados de assinatura de repositório MySQL.

Para executar a análise de vulnerabilidade, foi executado o seguinte código:

```
docker run -it –net host –pid host –userns host –cap-add audit_control \
    -e DOCKER_CONTENT_TRUST=$DOCKER_CONTENT_TRUST \
    -v /var/lib:/var/lib \
    -v /var/run/docker.sock:/var/run/docker.sock \
    -v /usr/lib/46ocker:/usr/lib/46ocker \
    -v /etc:/etc –label docker_bench_security \
```

```
docker/docker-bench-security
```

Após a conclusão do processo de verificação, o Docker Bench for Security exibirá um mini-relatório sobre as vulnerabilidades encontradas. A Figura 14 mostra o exemplo de relatório de análise de vulnerabilidade estática. A versão atual do Docker Bench for Security está no GitHub **(git clone** *https://github.com/Docker/Docker-bench-security.git***)** ou DockerHub (*Docker pull Docker/Docker-bench-security*).

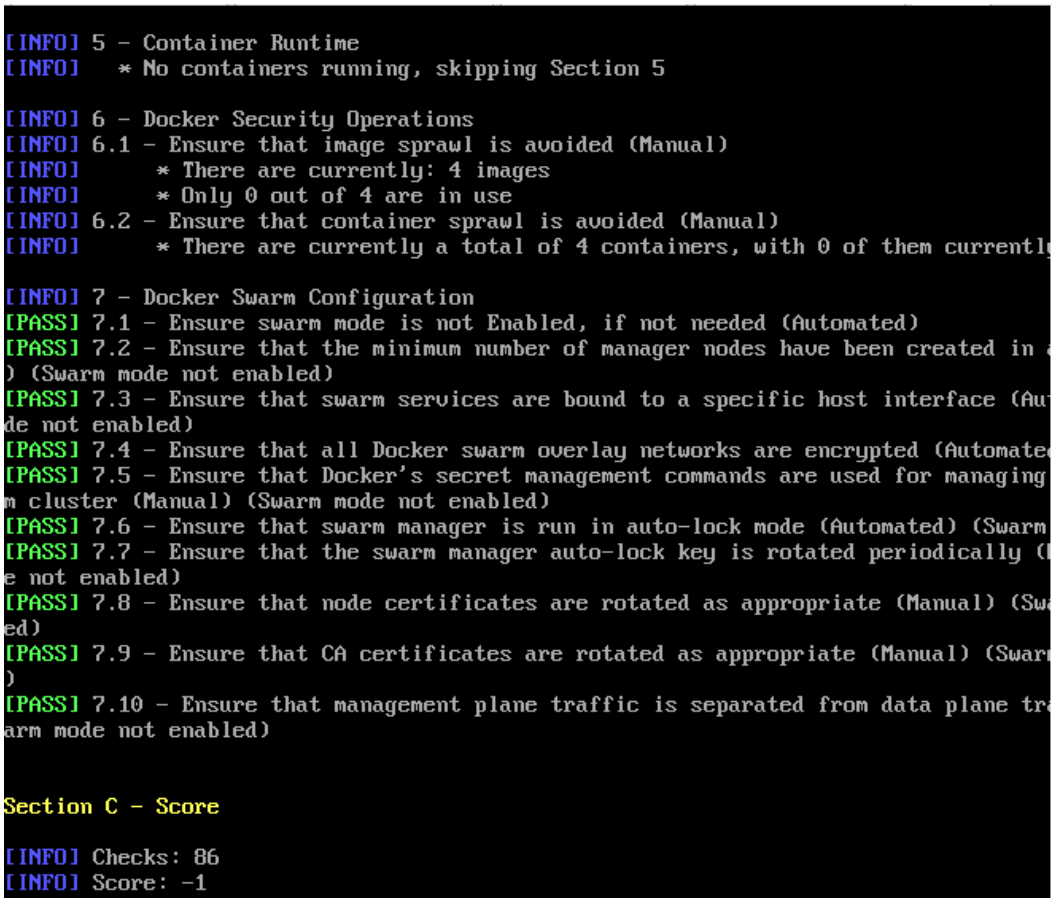

Figura 14. Mini-relatório do Docker Bench for Security

O subsistema de rede do Docker é conectado usando drivers. Vários drivers existem por padrão e fornecem funcionalidade básica de rede:

- **bridge**: driver de rede padrão. Se não especificar um driver, este é o tipo de rede que está criando. As redes em ponte geralmente são usadas quando os seus aplicativos são executados em containers independentes que precisam de se comunicar.
- **host**: para containers autônomos, removemos o isolamento de rede entre o container e o host do Docker e usa-se a rede do host diretamente.

- S**obreposição**: as redes de sobreposição conectam vários daemons do Docker e permitem que os serviços de enxame se comuniquem entre si. Também pode ser usar redes de sobreposição para facilitar a comunicação entre o serviço swarm e um container autônomo ou entre dois containers autônomos em diferentes daemons do Docker. Essa estratégia elimina a necessidade de roteamento entre esses containers no nível do SO.
- **ipvlan**: as redes Ipvlan dão aos utilizadores controle total sobre o endereçamento Ipv4 e Ipv6. O driver VLAN se baseia nisso, dando às operadoras controle total sobre a marcação VLAN da Camada 2 e até mesmo o roteamento Ipvlan L3 para utilizadores interessados em integração básica de rede.
- **macvlan**: As redes Macvlan permitem que se atribua um endereço MAC a um container, fazendo com que ele apareça como um dispositivo físico em sua rede. O daemon do Docker direciona o tráfego para containers com base nos seus endereços MAC. O uso do driver macvlan às vezes é a melhor escolha ao lidar com aplicativos herdados que exigem uma conexão direta com a rede física em vez de roteamento pela pilha de rede do host do Docker.
- **none**: desativa todas as conexões de rede para este container. Normalmente usado em conjunto com um driver de rede personalizado. Nenhum não está disponível para serviços de enxame.
- **Plug-ins de rede**: pode instalar e usar plug-ins de rede de terceiros com o Docker. Esses plug-ins estão disponíveis no Docker Hub ou em fornecedores terceirizados.

As redes de ponte definidas pelo utilizador são mais adequadas quando precisa de vários containers para se comunicar no mesmo host do Docker. As redes de host são mais adequadas quando a pilha de rede não precisa ser isolada do host do Docker, mas deseja que outros aspectos do container sejam isolados. As redes de sobreposição são mais adequadas quando precisa de containers executados em diferentes hosts do Docker para se comunicar ou quando vários aplicativos trabalham juntos usando serviços de enxame. As redes Macvlan são mais adequadas quando está migrando de uma configuração de máquina virtual ou precisa que seus containers se pareçam com hosts físicos em sua rede, cada um com um endereço MAC exclusivo. Plug-ins de rede de terceiros permitem integrar o Docker com pilhas de rede especializadas.

Para criar um canal ssh, utilizei o módulo Netmiko Python, baseado na conhecida biblioteca Paramiko para automação de rede de dispositivos e sistemas baseados em Linux.

4.2 Programação de algoritmos para encontrar anomalias de segurança em containers.

Para desenvolver o código do detetor de varredura de porta, precisamos realizar pesquisas para identificar marcadores que nos permitirão determinar a varredura de porta do host e dos containers individuais, bem como detetar o endereço IP da fonte de varredura.

Para desenvolver o código do detetor de varredura de porta, precisamos realizar pesquisas para identificar marcadores que nos permitirão determinar a varredura de porta do host e dos containers individuais, bem como detetar o endereço IP da fonte de varredura.

Foi utilizada a biblioteca Python Scapy como base para o futuro detetor de varredura de portas. Por exemplo, se abrir uma porta em um host Photon OS, onde o código sniffer Scapy deve ser executado primeiro. Então, após a digitalização, obteremos o resultado mostrado na Figura 15.

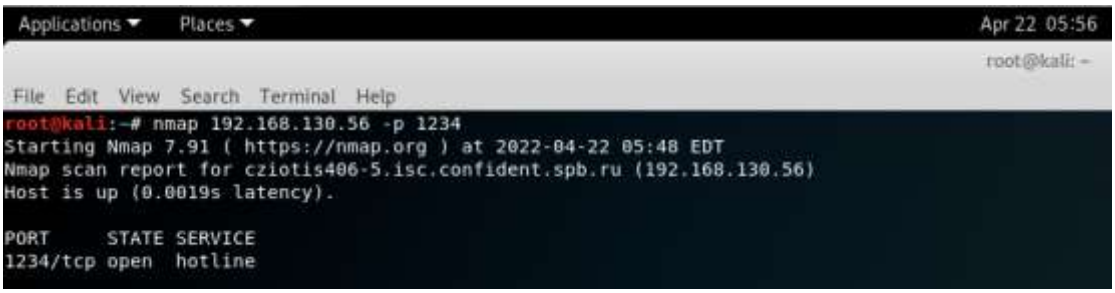

Figura 15.Port scanning em um host com containers.

O código a seguir começará a ouvir os pacotes:

```
def parsePacket(pkt):
    currentTime = datetime.now().strftime('%d-%m-%Y %H:%M')

    if IP in pkt:
        sourceAddr = pkt[IP].src
        destAddr = pkt[IP].dst
        if pkt[IP].dst=='192.168.130.56' and pkt[IP].src=='10.10.112.63'
and TCP in pkt:
            sourcePort = pkt[TCP].sport
            destPort = pkt[TCP].dport
            print('[{0}] [TCP] {1}:{2} -> {3}:{4}
{5}'.format(currentTime, sourceAddr, sourcePort, destAddr, destPort,
pkt[TCP].flags))

    return

while True:
    try:
        sniffer = sniff(lfilter=build_lfilter, count=0, prn=parsePacket)
        sys.exit()
    except socket.error:
        sys.exit()
```

Ao comparar o mapa de pacotes que vinham do scanner de porta e da aplicação legítima que transmitia o payload, determinei vários marcadores do scanner de porta:

- pacote ICMP de iniciante (79);
- falta ACK pacote depois de receber o pacote SYN ACK;
- em modo SYN scanner envia RST (90)

A Figura 16 mostra esta situação na maneira usual em Wireshark.

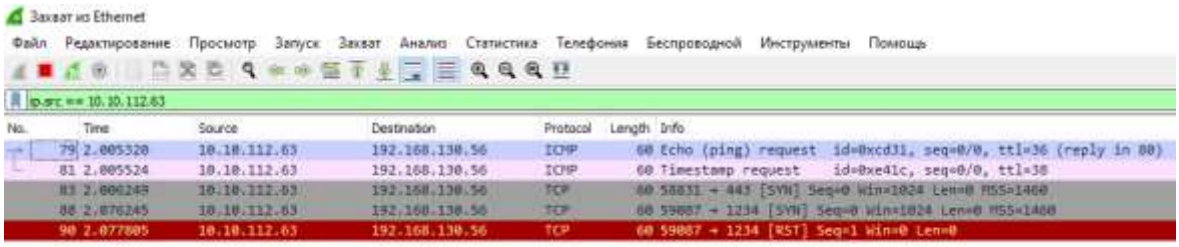

Figura 16. Captura dos pacotes do NMAP.

Se conhecendo os recursos de varredura de pacotes, podemos desenvolver um detetor para identificar um port scan do host. O princípio descrito acima para o nível do host também pode ser utilizado para identificar a varredura no nível do container.

Deteção de ARP spoofing no nível de pacotes de rede de container. Técnicas atuais de prevenção e deteção. Os métodos existentes para detectar a falsificação de ARP são discutidos a seguir na sequência.

Protocolo ARP Seguro (S-ARP). Este foi proposto como um substituto para o protocolo ARP. O protocolo S-ARP é definitivamente uma solução permanente para falsificação de ARP, mas a maior desvantagem é que teremos que fazer alterações na pilha de rede de todos os hosts. Não é muito escalável, pois atualiza a pilha para todos os sistemas operacionais disponíveis - algo que nem os fornecedores nem os clientes querem ficar felizes. Como o S-ARP usa um algoritmo de assinatura digital (DSA), temos uma sobrecarga adicional de computação criptográfica, embora os autores dos documentos argumentam que essas despesas gerais não são significativas. Entradas MAC estáticas. Adicionar endereços MAC estáticos por host para todos os outros hosts não permitirá falsificação, mas não é uma solução escalável e gerenciar todas essas entradas próprias emprego a tempo inteiro. Isso pode falhar miseravelmente se hosts móveis, como laptops, forem introduzidos

intermitentemente na rede. Também alguns sistemas operacionais eles são conhecidos por sobrescrever entradas ARP estáticas se receberem pacotes ARP Gratuitos.

Como resultado da aplicação dos métodos descritos acima, foi criado um código para um detetor ARP simples baseado em Scapy, conforme se apresenta na Figura 17.

```
from scapy.all import sniff

IP_MAC_Map = {}

def processPacket(packet):
    src_IP = packet['ARP'].psrc
    src_MAC = packet['Ether'].src
    if src_MAC in IP_MAC_Map.keys():
        if IP_MAC_Map[src_MAC] != src_IP:
            try:
                old_IP = IP_MAC_Map[src_MAC]
            except:
                old_IP = "unknown"
            message = ("\n Possível ataque ARP detectado \n "
                       + "É possível que a máquina com endereço IP \n "
                       + str(old_IP) + " está fingindo ser " + str(src_IP)
                       + "\n ")
            return message
        else:
            IP_MAC_Map[src_MAC] = src_IP


sniff(count=0, filter="arp", store=0, prn=processPacket)
```

Figura 17. A codificação de detetor de ARP spoofing.

Agora que temos código para detetar ataques e alguns problemas de segurança de container, podemos passar a combinar as funções criadas em um único bloco de backend. O framework Django foi escolhido para criar o backend devido à sua simplicidade e funcionalidade.

Uma das arquiteturas de aplicativos mais comuns – adotada tanto por amadores quanto por corporações – é o padrão Model-View-Controller (MVC), pois fornece uma separação clara de tarefas e responsabilidades entre os aspetos proeminentes de uma aplicação.

O Django segue vagamente este modelo. Em última análise, a separação de tarefas em grupos distintos serve a alguns propósitos diferentes. Diferentes camadas de visão e controlador podem se conectar a uma única camada de modelo. Isso permite que uma variedade de aplicativos compartilhem a mesma lógica e dados de negócios, apresentando-os e interagindo com eles de maneiras diferentes, para públicos diferentes.

Após a implantação do Django, temos uma estrutura de diretórios da nossa aplicação web, mostrada na Figura 18.

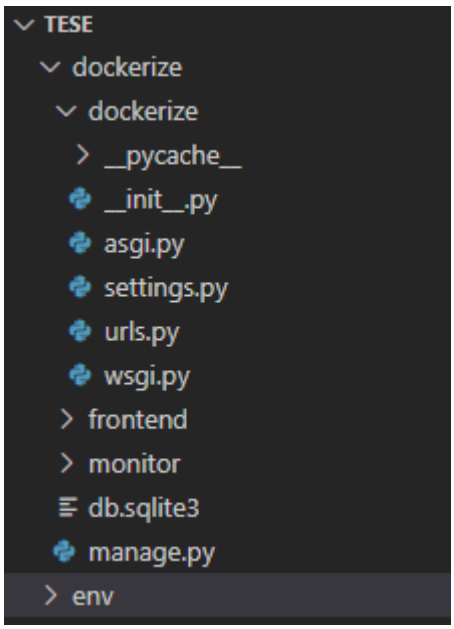

Figura 18. Estrutura das pastas da minha aplicação

Para Frontend foi utilizado o React. Como podemos ver na Figura 19, as pastas base são chamadas de "Monitor" que é uma aplicação Django e "Frontend" que é uma aplicação React. Esses nomes são fáceis de associar ao propósito que as respetivas pastas atendem. A Figura 19 mostra um dos principais ficheiros Python do Django, cujo código é projetado para implementar redireccionamentos de URI para a aplicação Monitor.

Figura 19. Conteúdo do ficheiro urls.py

Como podemos ver, no spreader base do Django, incluindo urls.py da aplicação “Monitor”.

4.3. Montagem de todos os elementos na versão alfa.

Para criar um frontend funcional foi escolhida uma biblioteca de diagramas baseada no trabalho do quadro React. Depois de alguma pesquisa foi escolhido o Recharts.

Os dados nos gráficos de Recharts são atualizados dinamicamente à medida que os objetos na matriz mudam. Isso pode ser feito, por exemplo, anexando uma +variável ao objeto, cujo valor será alterado (Figura 20).

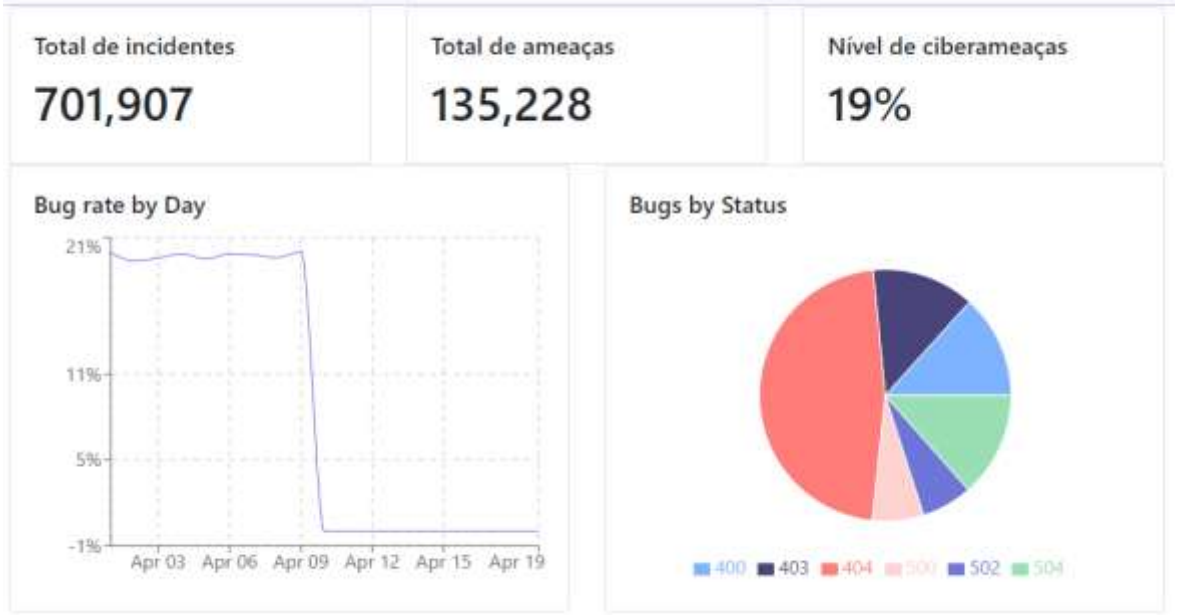

Figura 20. O gráfico de ciberameaças com ajuda Recharts

Os dados nos gráficos de Recharts são atualizados dinamicamente à medida que os objetos na matriz mudam. Isso pode ser feito, por exemplo, anexando uma +variável ao objeto, cujo valor será alterado.

Vários recursos interativos também podem ser adicionados aos gráficos. Por exemplo, passar o *rato* sobre um valor pode exibir uma dica de ferramenta. Pode conter texto definido para utilizadores. D3.js é uma biblioteca JavaScript para exibir dados digitais em um formato gráfico dinâmico. O D3 ajuda a dar vida aos dados com HTML, SVG e CSS. O D3 oferece controle total sobre o resultado visual final e é a tecnologia de visualização de dados mais popular e poderosa da web atualmente. De certa forma, o D3 é uma biblioteca JavaScript especializada que permite criar visualizações de dados incríveis usando uma abordagem mais simples (orientada a dados) usando padrões da web existentes. A biblioteca JavaScript D3 é muito auto-suficiente. Não depende de nenhuma biblioteca JavaScript que não seja a

que seu navegador já fornece. Na verdade, ele pode até ser utilizado em um ambiente sem navegador como o Node.js com configuração mínima. A maioria dos aplicativos da Web atuais funciona como um aplicativo de página única, construído com a parte frontal separada da parte traseira. Siga a arquitetura de microserviços. O backend geralmente é dividido em vários serviços. Normalmente, o backend do Cube.js é executado como um serviço que gere a conexão com o banco de dados, incluindo filas de consulta, armazenamento em cache e pré agregação. Ao mesmo tempo, a API é fornecida aos aplicativos front end para a criação de painéis e outras funções de análise conforme se esquematiza na Figura 21.

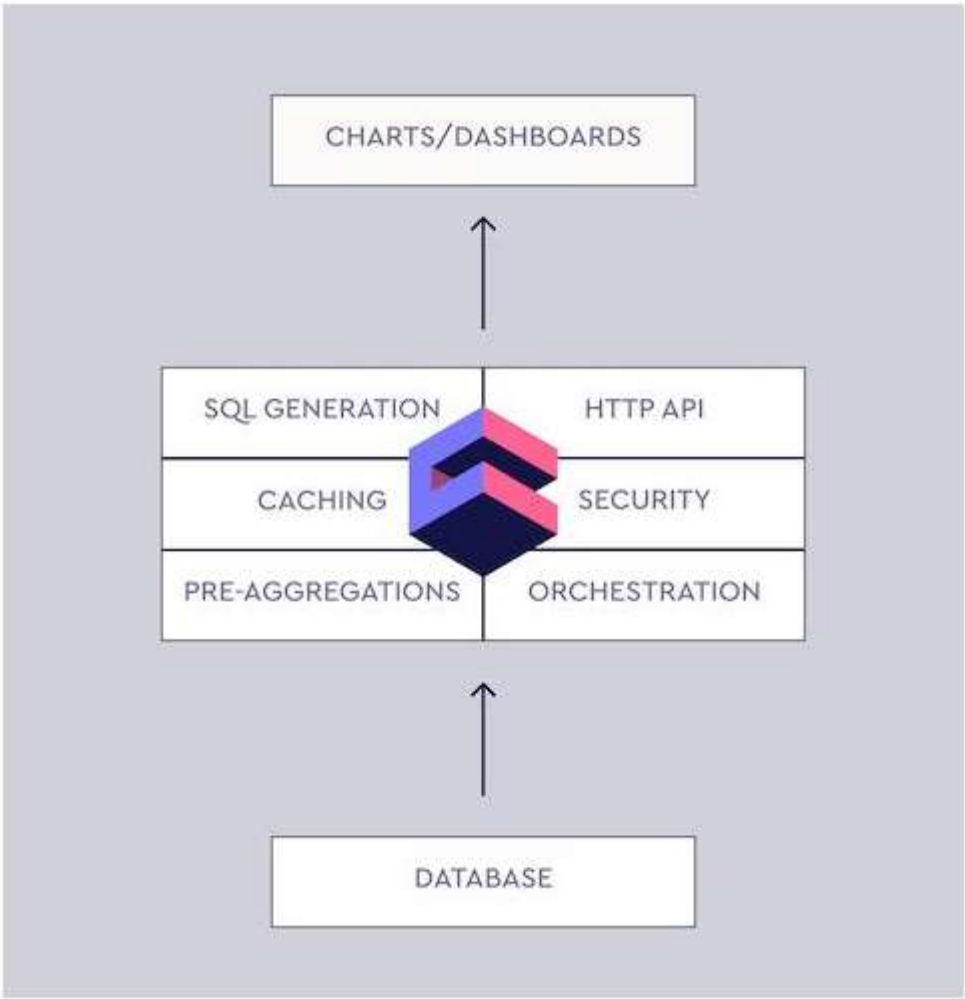

Figura 21. Esquema de movimentação de dados para visualização

Cube.js é uma plataforma de código aberto para a construção de aplicações web analíticos, que é usado principalmente para criar ferramentas internas de inteligência de negócios ou adicionar análises voltadas para o cliente a aplicações existentes. Na maioria dos casos, a primeira etapa na criação de tal aplicativo é analisar o dashboard. Ao começar a usar o Cube.js, deseja criar uma ferramenta que seja simples no início, mas facilmente

extensível em termos de funcionalidade, complexidade e volume de dados. O Cube.js estabelece uma base sólida para futuros sistemas de análise, seja um aplicativo independente ou integrado a um sistema de análise existente.

Nesta altura foi necessário cuidar da autenticação e criar um sistema de autenticação com tecnologia de servidor e gerir o fluxo de autenticação com tecnologia frontend. O Django vem com um modelo de sistema de autenticação integrado que se encaixa na maioria dos casos de utilizador e é bastante seguro. Mas foi necessário reescrevê-lo um pouco para atender às necessidades do meu projeto. Começámos por criar um modelo de utilizador personalizado estendendo o AbstractBaseUser.

O modelo de utilizador personalizado é um novo utilizador que herda de AbstractBaseUser. Mas também foi reescrito o CustomUser para configurar a criação de utilizadores no banco de dados e especificar isso no ficheiro de configurações *settings.py*:

```
# Points to the custom user model
AUTH_USER_MODEL = 'users.CustomUser'
```

Figura 22. Reescrita do CustomUser

Adicionar o conjunto de visualizações do utilizador (viewsets) foi o passo seguinte. Viewset é um conjunto de visualizações baseada em classe, capaz de lidar com todas as solicitações HTTP básicas: GET, POST, PUT, DELETE sem codificar nenhuma lógica. Ou se tiver necessidades específicas, poderá substituir esses métodos.

A estrutura REST fornece vários esquemas de autenticação prontos para uso, mas também posso implementar meus próprios esquemas. Eu usei autenticação com tokens JWT. Para isso, foi utilizado o *djangorestframework-simplejwt* para implementar a lógica de acesso/atualizaçãoadicionando *rest_framework_simplejwt.authentication.JWTAuthentication* à lista de classes de autenticação em settings.py:

```
REST_FRAMEWORK = {
    'DEFAULT_AUTHENTICATION_CLASSES': (
        'rest_framework_simplejwt.authentication.JWTAuthentication',
    ),
    'DEFAULT_RENDERER_CLASSES': (
        'rest_framework.renderers.JSONRenderer',
    )
}
```

Figura 23. Estrutura de REST

A biblioteca Simple JWT vem com duas rotas úteis:

- Um para obter o token de acesso e atualizar (login) 'api/token/'
- E outro para obter um novo token de acesso usando o token de atualização 'api/token/refresh/'

Durante o processo de registro do utilizador, o utilizador deverá fazer login novamente para obter um token. Foi testada a API com ajuda o Postman (Figura 25 e 26).

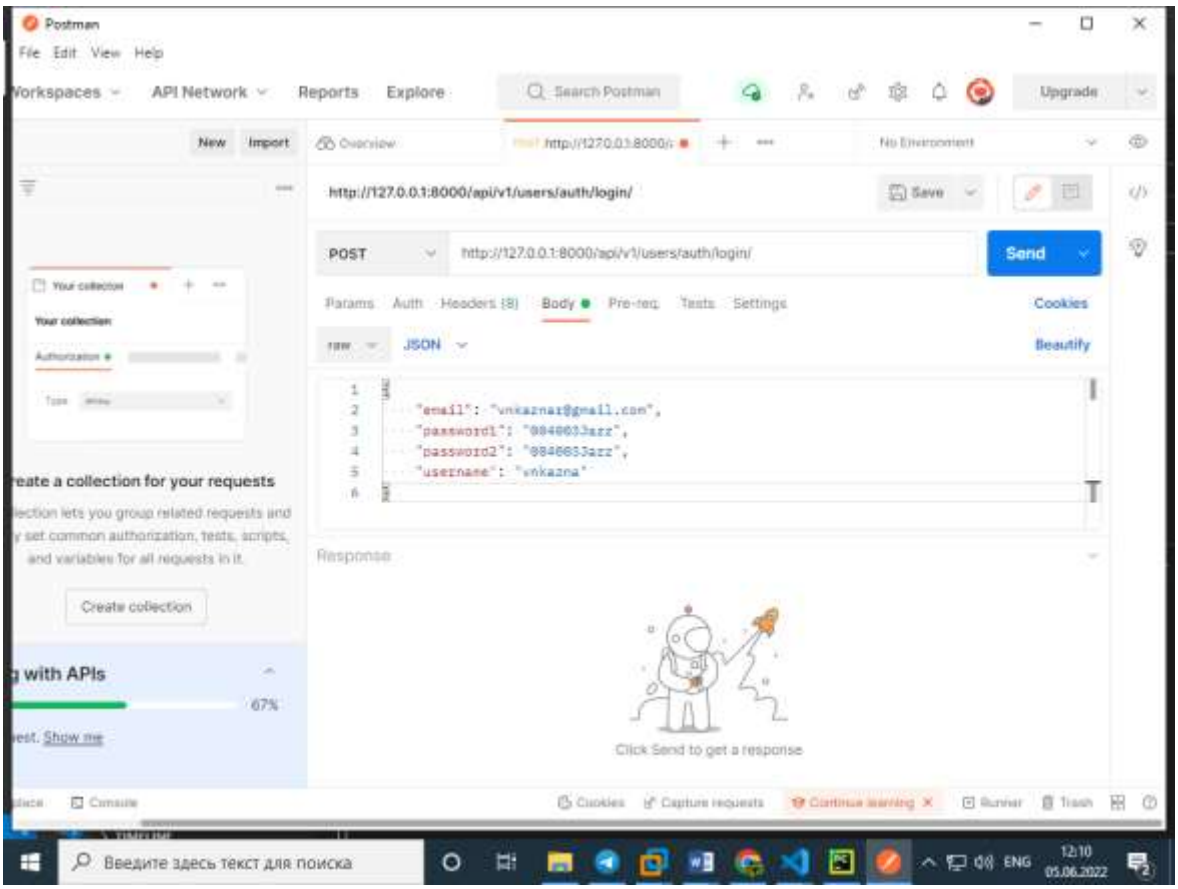

Figura 24. POST- req para teste de autenticação em DRF API

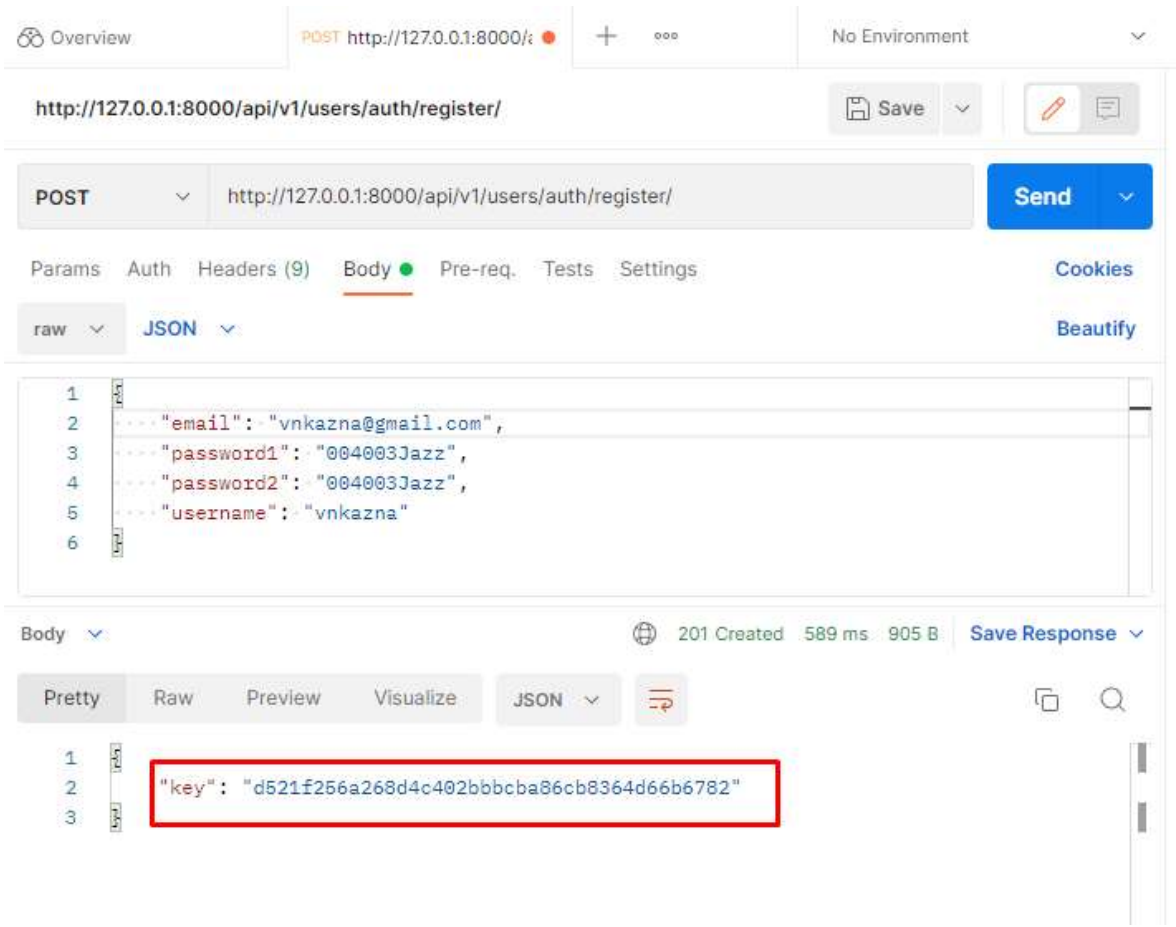

Figura 25. Obtenção do token de autorização com ajuda o Postman.

Com todos esses arquivos criados, agora podemos registrar e autorizar utilizadores com nossos modelos personalizados do Django e autenticar com sucesso os utilizadores com JSON Web Tokens. Embora a maioria dessas informações tenha sido postada nesta página, espero que tenha ajudado aqueles que desejam fazer algo semelhante. Agora posso testar o formulário de autenticação do utilizador do sistema.

Por padrão, o Django usa sessões para autenticação. Antes de prosseguirmos, julgamos que devemos referir sobre o que isso significa, por que é importante, o que é autenticação baseada em token e os que é JSON Web Token Authentication (JWT abreviado).

No Django, as sessões são armazenadas em cookies. Essas sessões, juntamente com middlewares integrados e objetos de solicitação, garantem que o utilizador esteja disponível em todas as solicitações. O utilizador pode ser acedido como *request.user*. Quando um utilizador está conectado, request.user é uma instância da classe User. Quando ele efetua logout, request.user é uma instância da classe AnonymousUser. Esteja o utilizador autenticado ou não, *request.user* sempre existirá. Quando é necessário saber se o utilizador

atual está autenticado, pode usar *request.user.isauthenticated()* que retornará True se o utilizador estiver autenticado e False caso contrário. Se request.user for AnonymousUser, *request.user.isauthenticated()* retornará False. Isso permite que converta se request.user não for None e request.user.isauthenticated(): in if request.user.isauthenticated():

No nosso caso, o cliente e o servidor funcionam em locais diferentes. Por exemplo, o servidor será iniciado em http://localhost:3000 e o cliente em http://localhost:5000.

A alternativa mais comum à autenticação baseada em sessão/sessão é a chamada. autenticação baseada em token. Usámos uma forma especial dessa autenticação para proteger o nosso aplicativo. Com a autenticação baseada em token, o servidor fornece ao cliente um token após uma solicitação de login bem-sucedida. Este token é único para o utilizador e é armazenado no banco de dados junto com o ID do utilizador (para ser mais preciso, são possíveis variações antecipadas da geração do token, a ideia principal é que ele autentique o utilizador, permitindo que saiba quem é e dando acesso à api, e tinha uma vida útil, de acordo com a expiração do qual "podre"). Espera-se que o cliente envie o token junto com solicitações futuras para que o servidor possa identificar o utilizador. O servidor faz isso procurando uma tabela de banco de dados contendo todos os tokens gerados. Se um token correspondente for encontrado, o servidor continuará verificando se o token é válido. Se não for encontrado, dizemos que o utilizador não está autenticado. Como os tokens são armazenados no banco de dados e não em cookies, a autenticação baseada em token atende às minhas nece JSON Web Token (abreviatura JWT) é um padrão aberto (RFC 7519) que define uma forma compacta e independente de transferir informações com segurança entre duas partes.

Ao mudar de tokens regulares para JWT, obtivemos várias vantagens:

- JWT é um padrão aberto. Isso significa que todas as implementações de JWT devem ser bastante semelhantes, o que é uma vantagem ao trabalhar em diferentes linguagens e tecnologias. Os tokens regulares têm uma forma mais livre, o que permite ao desenvolvedor decidir a melhor forma de implementar os tokens.
- Os JWTs contêm informações sobre o utilizador, o que é conveniente para o lado do cliente.
- As bibliotecas assumem a maior parte do trabalho duro aqui. Implantar seu próprio sistema de autenticação é perigoso, então deixamos as coisas importantes para bibliotecas testadas nas quais podemos confiar.

Agora podemos testar o formulário de autenticação do utilizador do sistema (Figura 27).

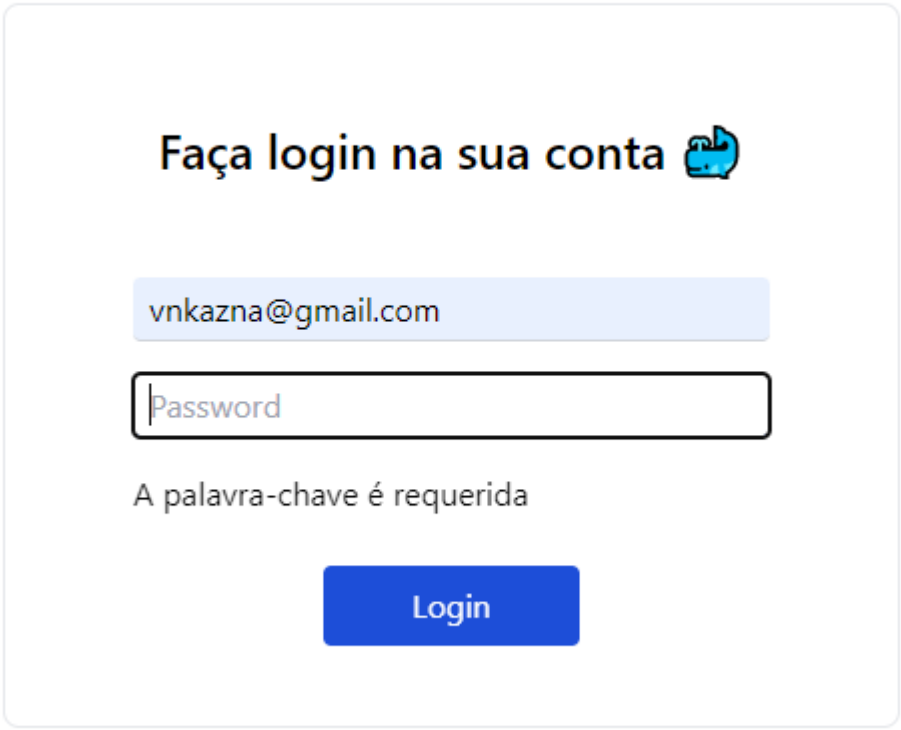

Figura 26. A forma de autenticação para utilizadores

O Django tem a ideia de backends de autenticação. O back-end é essencialmente um plano para decidir se o utilizador é autenticado (Figura 28). Precisamos criar nosso próprio backend para suportar JWT, pois nem o Django nem o Django REST Framework (DRF) o suportam por padrão.

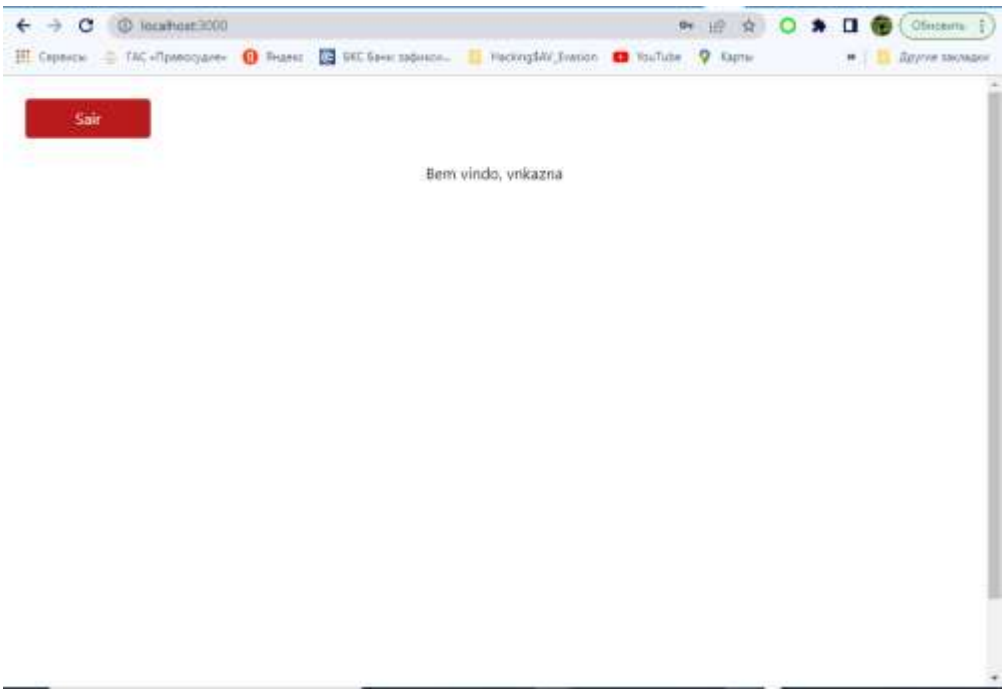

Figura 27. Login com sucesso.

4.4. Teste funcional e teste de produtividade.

Utilizei os vários tipos de testes de produtividade era preciso as cargas representativas do sistema devem ser modeladas, geradas e apresentadas ao sistema em teste. As cargas são comparáveis às entradas usadas para casos de teste funcionais, mas diferem das seguintes maneiras principais:

- **Geração de carga através da interface do utilizador.** Essa pode ser uma abordagem adequada se apenas um pequeno número de utilizadores for representado e se houver o número necessário de clientes de software para fornecer a entrada necessária. Essa abordagem também pode ser usada em conjunto com ferramentas de execução de testes funcionais, mas pode rapidamente se tornar impraticável à medida que o número de utilizadores simulados aumenta. A estabilidade da interface do utilizador (UI) também é uma dependência crítica. Mudanças frequentes podem afetar a repetibilidade dos testes de desempenho e podem afetar significativamente os custos de manutenção. O teste de interface do utilizador pode ser a abordagem mais representativa para testes de ponta a ponta.
- **Geração de carga via interface de programação de aplicativos (API).** Essa abordagem é semelhante ao uso de uma interface de utilizador para entrada de dados, mas usa uma API de aplicativo em vez de uma interface de utilizador para simular a

interação do utilizador com o sistema em teste. Assim, essa abordagem é menos sensível a alterações (como atrasos) na interface do utilizador e permite que as transações sejam processadas da mesma forma como se fossem inseridas diretamente pelo utilizador por meio da interface do utilizador. Criei os scripts dedicados que chamam repetidamente determinados procedimentos de API e permitem que mais utilizadors sejam simulados em comparação com o uso de entradas de interface do utilizador.

- **Geração de carga usando protocolos de comunicação capturados.** Essa abordagem envolve capturar as interações do utilizador com o sistema em teste no nível do protocolo de comunicação e, em seguida, reproduzir esses cenários para simular um número potencialmente muito grande de utilizadores de maneira repetível e confiável.

O desempenho do sistema não deve ser mais do que minimamente afetado pelo esforço para coletar métricas (conhecido como "efeito de sondagem"). Além disso, o volume, a precisão e a velocidade com que as métricas de desempenho devem ser coletadas tornam o uso da ferramenta imprescindível. Embora o compartilhamento de ferramentas não seja incomum, pode levar ao uso redundante de ferramentas de teste e outros problemas. Os efeitos que testei são:

- Alto uso de recursos, como alto uso de CPU, alto consumo de armazenamento em disco e largura de banda insuficiente
- Erros de memória e avisos, como esgotamento de memória
- Problemas de deadlocks e multithreading, especialmente ao realizar operações de banco de dados
- Os erros de banco de dados, como exceções SQL e tempos limite SQL.

Em testes funcionais, especialmente ao testar certos requisitos funcionais ou elementos funcionais de histórias de utilizadores, os resultados esperados geralmente podem ser claramente definidos e os resultados do teste interpretados para determinar se o teste foi aprovado ou reprovado. Por exemplo, o relatório de vendas mensal mostra o total correto ou o total incorreto.

Enquanto os testes que testam a adequação funcional geralmente se beneficiam de oráculos de teste bem definidos, os testes de desempenho geralmente carecem dessa fonte de informação. O teste de desempenho foi iterativo. Cada teste fornece informações valiosas sobre o desempenho do aplicativo e do sistema. As informações coletadas de um único teste

são usadas para corrigir ou otimizar os parâmetros do aplicativo e do sistema. A próxima iteração do teste mostrará os resultados das modificações e assim por diante, até que os objetivos do teste sejam atendidos. Os riscos de desempenho associados à virtualização incluem estresse excessivo de hardware em todas as máquinas virtuais ou configuração incorreta da máquina virtual host, resultando em escassez de recursos.

Utilizou-se o Paytest que tem a capacidade de conectar plugins. Esse recurso permite começar a usar um módulo como o Pytest-drf:

```
pip install pytest-drf
```

```
from django.contrib.auth.models import User
from django.urls import reverse
from pytest_drf import APIViewTest, AsUser, Returns200, UsesGetMethod
from pytest_lambda import lambda_fixture
alice = lambda_fixture(
    lambda: User.objects.create(
        username='vnkazna',
        first_name='Vladimir',
        last_name='Kaznacheev',
        email='vnkazna@gmail.com',
    ))

class TestAboutUs(
    APIViewTest,
    UsesGetMethod,
    Returns200,
    AsUser('vnkazna'),
):
    url = lambda_fixture(lambda: reverse('about-us'))

    def test_it_returns_profile(self, json):
        expected = {
            'username': 'vnkazna',
            'first_name': 'Vladimir',
            'last_name': 'Kaznacheev',
            'email': 'vnkazna@gmail.com',
```

4.5. Fixes de bugs.

Tendo criado uma lista de bugs (Figura 29), iniciou-se a primeira iteração da correção do bug.

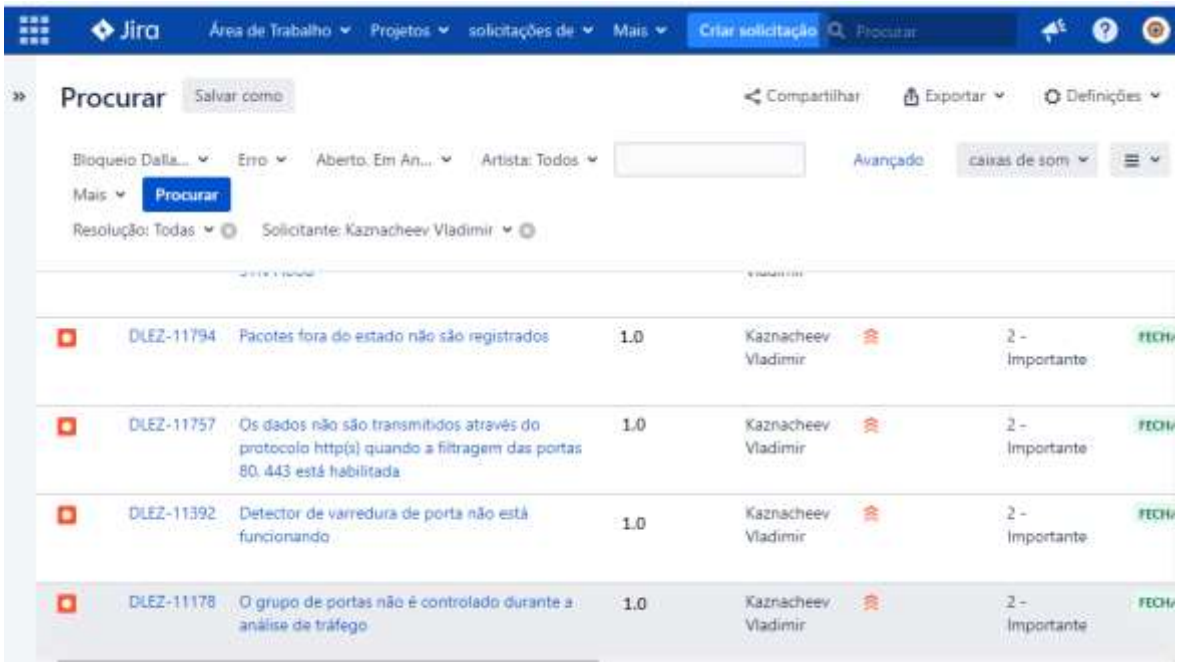

Figura 28. A lista de bugreports do protótipo.

As correções de bugs, bem como desenvolvimento e testes, foram realizados de acordo com o cronograma geral e o modelo iterativo adotado no início do Capítulo 3.

No mesmo estágio, habilitou-se o controle do código-fonte usando o Git e o GitHub. Este último também servirá como repositório público para testes de protótipos. Para fazer isso, inclui-se "SourceControl " nos IDEs. Para codificação, começou-se usando o PyCharm, e continuou-se no VSCode. Para o VSCode, habilitou-se o SourceControl usando o módulo de terceiros "GitHub Pull Requests and Issues extension".

DISCUSSÃO DOS RESULTADOS

5.1 Resultados mais relevantes

Os resultados deste trabalho consistem na criação de uma plataforma utilizando a linguagem de programação Python, nomeadamente o Django Framework, enquanto o frontend e a GUI de utilizadores utilizam a linguagem de programação Java Script, ou seja, o React Framework. Com este protótipo foram solucionados os problemas de avaliação da segurança da infraestrutura containerizada e do fornecedor cruzado das tecnologias utilizadas para a sua construção. Além disso, foram implementadas funções e visualizações seguindo as recomendações para aumentar o nível de segurança, fechando vulnerabilidades conhecidas e reduzindo a superfície de um possível ataque. Foram confirmados os problemas existentes na segurança das infraestruturas de contentores, comuns a todos os fornecedores e específicos a cada uma das tecnologias de contentorização.

O resultado da análise e desenvolvimento são as conclusões contidas nos métodos utilizados para controlar os principais parâmetros de segurança, bem como o código do protótipo, que está localizado no Github e o container Dockerhub, que se designou de container master. Essas ferramentas são um auxílio visual e uma demonstração de como, tendo conhecimento na interseção das disciplinas de cibersegurança e web design, pode garantir não apenas a segurança básica de uma nuvem privada, mas também fazer a sua monitorização.

Nesta investigação obtiveram-se resultados em duas dimensões que se completam:

- teóricos;
- práticos.

Em teoria, analisaram-se os trabalhos e pesquisas sobre este tema, o que permitiu definir e implementar novos métodos e ferramentas para monitorar os parâmetros de segurança de um ambiente de container.

Na dimensão prática, este trabalho apresenta um protótipo contendo a base de código de detetores e web serviços e interface para apresentar informações de suporte à tomada de decisão.

Durante o desenvolvimento do protótipo, deu-se ênfase especial aos testes e ao desenvolvimento de uma base e cobertura de testes nos ambientes de container que permitam tomar medidas para eliminar as ameaças.

A análise teórica mostrou como a segurança pode ser multifacetada em ambientes de container, que pode haver dezenas de vetores de ataque que ficam na superfície e, quando aprofundamos o problema, podemos encontrar centenas de ameaças e vetores. Garantir a segurança cibernética é um processo contínuo, responsável e meticuloso. A monitorização contínua dos três pilares de confidencialidade, integridade e disponibilidade de dados, bem como o desenvolvimento contínuo de competências na questão de possíveis vulnerabilidades, desempenha um papel importante na garantia da segurança dos dados.

Desta forma, utilizamos não só uma abordagem tecnológica à segurança, mas também uma abordagem intelectual, consubstanciada no crescimento contínuo das competências do pessoal técnico.

Os resultados da investigação podem ser aplicados para os diversas tarefas.

Em primeiro lugar, o trabalho pode servir como uma instrução aos Engenheiro de DevSecOps para construir o esqueleto de um sistema de monitoramento de segurança em nuvem privada. Além disso, os exemplos de código permitem que implemente exemplos específicos de como combinar estruturas mundialmente famosas para criar elementos de um sistema de segurança em uma empresa. Um resultado inesperado é que as empresas que têm um Engenheiro de DevSecOps em sua equipe não precisam gastar um orçamento na compra de software especializado para proteção de ambientes containerizados. Eles podem projetar seu próprio sistema de proteção com base em ferramentas de código aberto bem conhecidas e testadas ao longo do tempo.

5.2 Discussão dos resultados

A investigação mostra que a monitorização de segurança em tempo real é mais eficaz para trazer a segurança dos sistemas de informação a um nível aceitável do que a monitorização periódica e o uso das configurações fornecidas pelas plataformas/sistemas standard.

Comparando os resultados deste estudo com os resultados dos trabalhos apresentados na revisão de literatura, algumas conclusões podem ser tiradas. Por exemplo, o estudo mais relevante de Sadique e Cassell (2021) confirmou a importância da visualização de dados.

Nesta investigação, eles descreveram os principais objetivos de usabilidade da interface para visualizar efetivamente as ameaças nas visualizações da rede. Em seguida, eles descrevem sua metodologia para investigar propriedades visuais eficazes de gráficos que facilitam a identificação de ameaças. Os resultados qualitativos mostraram que alguns profissionais de segurança cibernética tinham dúvidas, confiando apenas nas propriedades do gráfico visual. (Farhan Sadique, 2021). No meu estudo, também confirmei a presença de um grande número de vetores de ataque e os benefícios da visualização.

O meu estudo mostra-se mais completo e detalhado, descrevendo o processo tecnológico de criação de um sistema de monitorização de containers na junção de tecnologias web e cibersegurança. A semelhança, por sua vez, está no fato de que a mesma abordagem foi utilizada para garantir a segurança das infraestruturas de containers.

5.3 Limitações do trabalho

Certamente o trabalho precisa de ser continuado. Em primeiro lugar, é necessário continuar a trabalhar na identificação de uns novos vetores de ataque. Também é necessário expandir e detalhar a funcionalidade do painel. A terceira prioridade no contexto de trabalho contínuo é construir a base de código de sensores e detetores, bem como aplicações web únicas que permitem explorar ferramentas previamente desenvolvidas.

O principal fator limitativo para o desenvolvimento deste trabalho consistiu na dificuldade na criação de uma forma confiável de embutir analisadores e detetores no código dos serviços web.

CONCLUSÕES

Em conclusão, gostaria de resumir os resultados de todas as pesquisas, bem como os resultados aplicados e seu impacto na segurança adequada de dados e infraestrutura comerciais. Novos desafios e problemas de cibersegurança nos obrigam a pesquisar as novas formas de proteger os dados. Entre eles está a compra de novas ferramentas de desenvolvedores conhecidos neste campo. No entanto, também pode considerar desenvolver seu próprio sistema de monitoramento, que posteriormente pode ser integrado ao CI/CD e servir como um contador de ameaças para o DevSecOps.

Como resultado da pesquisa, respondemos de forma inequívoca à questão principal sobre a automatização da recolha de dados relacionadas à segurança cibernética de containers.

Com a ajuda do uso de ferramentas próprias desenvolvidas e ferramentas open source comprovadas, uma resposta positiva também foi dada à visualização dos dados coletados.

O trabalho pode continuar a ser desenvolvido em várias direções:

- como um projeto de segurança interna (é necessário suporte constante e dispêndio de recursos intelectuais e computacionais).
- como startup (é necessário investimento em infraestrutura e marketing).
- como um guia para unir tecnologia da web e segurança cibernética (é necessária uma mudança mínima para cada indústria).

BIBLIOGRAFIA

Abdelrazik, A. (2019). Introduction to Photon OS. Palo Alto, California, USA.

Ahmad Imtiaz, S. S. (2019). Container Security: Issues, Challenges, and the Road Ahead. *IEEE Access*.

Ahtisham Hashmi, A. R. (2018). *Security and Compliance Management in Cloud.* Greater Noida: School of Computing Science and Engineering.

Alchin, M. (2008). *Pro Django.* Apress.

Andreas Aspernäs, M. N. (2016). *Container Hosts as Virtual Machines.* Växjö: Bachelor Thesis Project.

Angela Orebaugh, B. P. (2018). *Nmap in the enterprise. Your guide to Network Scaning.* Syngress Publishing, Inc.

Antonino Sabetta, M. B. (2018, September). IEEE International Conference on Software Maintenance and Evolution (ICSME). *A Practical Approach to the Automatic Classification of Security-Relevant Commits.* Madrid: IEEE.

Babak Bashari Rad, H. J. (2017). An Introduction to Docker and Analysis of its Performance. *IJCSNS International Journal of Computer Science and Network Security,* VOL.17.

Belitz, P. (2020, August 7). Container Image Signatures in Kubernetes. *SSE—Secure Systems Engineering*.

BEN-YAACOV, G. (2018). Verification and Validation for Advanced Train Control Systems.

Bosco, J. P. (2020). *Docker como ferramenta para construção de um ambiente em nuvem.* São Paolo: FACULDADE DE TECNOLOGIA DE AMERICANA.

By Neil MacDonald, C. W. (2021). Innovation Insight for Cloud-Native Application Protection Platforms. *Gartner*.

Byungchul Tak, C. I. (2017). Understanding Security Implications of Using Containers in the Cloud.

Cervone, H. F. (2011, February 15). Understanding agile project management methods using Scrum. *OCLC Systems & Services: International digital library perspectives*.

Chadni Islam, M. A. (2020). *A Multi-Vocal Review of Security Orchestration.* Canberra: CREST – The Centre for Research on Engineering Software.

Collins, M. (2020). *Network Security. Through Data Analysis.* O'Reilly.

Dennis Jerome, J. K. (2021). Platforms and Technologies. *ATARC Application Development Working Group*.

Dharmesta, P. A. (2020). Effectiveness of Sniffer Using Natural Language in Learning Computer Network Traffic. *Jurnal RESTI*.

Díaz, J., & Pérez, J. E. (2019). Self-Service Cybersecurity Monitoring as Enabler for DevSecOps. *IEEE Xplore*.

Eder, M. (2016). Seminar Future Internet WS2015/16. *Hypervisor- vs. Container-based Virtualization.* München: Fakultät für Informatik.

Eliane Figueiredo Collins, V. F. (2012). 7th International Workshop on Automation of Software Test (AST). *Software Test Automation Practices in Agile Development Environment: An Industry Experience Report* . Zurich: IEEE.

Flournoy, D. (2021). Overview: Satellite Signal Security: Cop view: Satellite Signal Security. *Online Journal of Space Communication*.

Gackenheimer, C. (2015). *Introduction to React.* Apress.

Harris, C. (2021). *Agile scrum artifacts.* Retrieved from https://www.atlassian.com/agile/scrum/artifacts

Hausenblas, M. (2018). *Container Networking From Docker to Kubernetes.* Beijing: O'Reilly.

Ikhwan, M. (2021). *ANÁLISE DA IMPLEMENTAÇÃO DE GESTÃO UNIFICADA DE AMEAÇAS (UTM) PARA SEGURAR KUBERNETES.* Mataram: BUMIGORA.

Jiang Wenhao, L. Z. (2021). IEEE 3rd International Conference on Information Systems and Computer Aided Education (ICISCAE). *Vulnerability Analysis and Security Research of Docker Container.* Dalian: IEEE.

Jun-Young Park, E.-N. H. (2020). A Cost-Optimization Scheme Using Security. *Journal of Information Processing System*.

Kaimer, M. (2020). *bWAPP vs. Juice Shop Unsichere Webservices.*

Kelly Brady, S. M. (2020). 10th Annual Computing and Communication Workshop and Conference (CCWC). *Docker Container Security in Cloud Computing.* Las Vegas: IEEE.

Ken Schwaber, J. S. (2020, Novembro). *scrumguides.org.* Retrieved from scrum.org: https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-US.pdf#zoom=100

Kumar, S. A. (2016). IEEE International Conference on Services Computing (SCC). *Securing Docker Containers from Denial of Service (DoS) Attacks.* San Francisco: IEEE.

Menouer, T. (2021). KCSS: Kubernetes container scheduling strategy. *The Journal of Supercomputing volume*.

Michael Falk, O. H. (2017). *Static Vulnerability Analysis of Docker Images.* Karlskrona: BTH.

Michael Wurster, U. B. (2020). The essential deployment metamodel: a systematic review of deployment automation technologies. *SICS Software-Intensive Cyber-Physical Systems*.

Miguel Borges de Freitas, P. Q. (2020). IEEE/IFIP Network Operations and Management Symposium. *SDN-assisted containerized security and monitoring components.* Budapest: IEEE.

Miniman, S. (2015). Infrastructure & Cloud. *Wikibon.*

Müller, J. N. (2020). *Static Source Code Analysis of Container*. Munich: TECHNICAL UNIVERSITY OF MUNICH.

Nicola Garzaniti, S. B. (2019). 2019 IEEE Aerospace Conference. *Effectiveness of the Scrum Methodology for Agile Development of Space Hardware.* Big Sky: IEEE .

Nikolas Jensen, C. C. (2021). XXI SIMPÓSIO BRASILEIRO EM SEGURANÇA DA INFORMAÇÃO E DE SISTEMAS COMPUTACIONAIS. *Análise de segurança dos orquestradores Kubernetes, Docker Swarm e Apache Mesos baseada no CVE / MITRE.* Santa Catarina: Departamento de Ciencia da Computação.

Okken, B. (2017). *Python Testing with pytest.* The Pragmatic Programmers, LLC.

Olsen, K. (2018). *Certified Tester Foundation Level Syllabus.* ISTQB.

Olsen, K. (2018). *Foundation Level Syllabus.* ISTQB.

Olufogorehan Tunde-Onadele, J. H. (2020). IEEE International Conference on Cloud Engineering (IC2E). *A Study on Container Vulnerability Exploit Detection.* Prague: IEEE .

Omar Javed, S. T. (2021). *Understanding the Quality of Container Security Vulnerability Detection Tools.* Lugano: Universita della svizzera italiana.

Ovaskainen, L. (2021). *IMPLEMENTAÇÃO DA FERRAMENTA DE PREVISÃO POR REACT.* Helsinki: Escadas Digitais.

Panagiotis, M. (2020). *Attack methods and defenses on Kubernetes.* Piraeus: MSC DIGITAL SYSTEMS SECURITY.

Paul MIHĂILĂ, T. B. (2017). MACRo 2017 - 6th International Conference on Recent Achievements in Mechatronics, Automation, Computer Science and Robotics. *Network Automation and Abstraction using Python Programming Methods.* Brașov: ”Transilvania” University.

Raheem, M. (2021). *Implementing a Secured Container Workload in the Cloud.* Kouvola: South-Eastern Finland University of Applied Sciences.

René Peinl, F. H. (2016). Docker Cluster Management for the Cloud - Survey Results and Own Solution. *Journal of Grid Computing*.

Rice, L. (2020). *Containe Security.* Boston: O'Reilly.

Rita Marques, M. M. (2018). IEEE 20th Conference on Business Informatics (CBI). *Assessing Agile Software Development Processes with Process Mining: A Case Study.* Vienna: IEEE.

Sarycheva Yulia Yurievna, L. S. (2020). Testes Automatizados ds API. *StudNet*.

Sergey Belov, J. J. (2020). 27th International Symposium Nuclear Electronics and Computing (NEC’2019). *Big data technologies for labour market analysis.* Budva: NEC 2019.

Shaila R Ghanti, G. (2015). Design of System on Chip for Generating SYN Flood Attack to Test the Performance of the Security System. *International Journal of Computer Applications*.

Si Liao, C. Z. (2020). International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC). *A Comprehensive Detection Approach of Nmap: Principles, Rules and Experiments.* Chongqing: IEEE.

Sliger, M. (2011). PMI® Global Congress 2011. *Agile project management with Scrum.* Dallas: Project Management Institute.

Smith, R. (2017). *Docker Orchestraition.* Birmingham: Packt Publishing.

Syed, M. H. (2019). *Modeling and Security in Cloud and Related Ecosystems.* Boca Raton: ProQuest Dissertations Publishing.

Thomas Watts, R. B. (2019). Hawaii International Conference on System Sciences. *Insight from a Docker Container Introspection.* Honolulu.

Vivek Ramachandran, S. N. (2005). International Conference on Information Systems Security. *Detecting ARP Spoofing: An Active Technique.* Indian Institute of Technology Guwahati.

Whiting, E. (2018). *Performance Testing and Agile Software.* Clear Lake: College of Science and Engineering, University of Houston.

Xiaozhou Li, S. M. (2022). *Exploring Factors and Metrics to Select Open Source.* Tampere.

Xin Lin, L. L. (n.d.). ACSAC '18: Proceedings of the 34th Annual Computer Security Applications. *A Measurement Study on Linux Container Security: Attacks and Countermeasures.*

Zhecho D. Mitev, J. R.-L. (2019). *Distributed Port Scanning.* Maastricht: Maastricht University.

Zhu, N. Q. (2013). *Data Visualization with D3.js Cookbook.* Birmingham: Packt Publishing.