UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
Câmpus de Rio Claro – SP

Ronaldo Rodrigues Martins

Autoproteção para a camada de aplicação: uma abordagem baseada em técnicas de aprendizado e no laço de controle MAPE-K

Rio Claro – SP
2022

Ronaldo Rodrigues Martins

Autoproteção para a camada de aplicação: uma abordagem baseada em técnicas de aprendizado e no laço de controle MAPE-K

Dissertação apresentada como parte dos requisitos para obtenção do título de Mestre em Ciência da Computação, junto ao Programa de Pós-Graduação em Ciência da Computação, do Instituto de Geociências e Ciências Exatas da Universidade Estadual Paulista "Júlio de Mesquita Filho", Câmpus de Rio Claro – SP.

Orientador: Prof. Dr. Frank José Affonso

Rio Claro – SP

2022

M386a

Martins, Ronaldo Rodrigues
Autoproteção para a camada de aplicação : Uma abordagem baseada em técnicas de aprendizado e no laço de controle MAPE-K / Ronaldo Rodrigues Martins. -- Rio Claro, 2022
151 p.

Dissertação (mestrado) - Universidade Estadual Paulista (Unesp), Instituto de Geociências e Ciências Exatas, Rio Claro
Orientador: Frank José Affonso

1. Autoproteção. 2. Sistemas autoadaptativos. 3. Ciência de dados. 4. Cibersegurança. 5. Modelos de classificaçã. I. Título.

Sistema de geração automática de fichas catalográficas da Unesp. Biblioteca do Instituto de Geociências e Ciências Exatas, Rio Claro. Dados fornecidos pelo autor(a).

Essa ficha não pode ser modificada.

Ronaldo Rodrigues Martins

Autoproteção para a camada de aplicação: uma abordagem baseada em técnicas de aprendizado e no laço de controle MAPE-K

Dissertação apresentada como parte dos requisitos para obtenção do título de Mestre em Ciência da Computação, junto ao Programa de Pós-Graduação em Ciência da Computação, do Instituto de Geociências e Ciências Exatas da Universidade Estadual Paulista "Júlio de Mesquita Filho", Câmpus de Rio Claro – SP.

Comissão Examinadora

- Prof. Dr. Frank José Affonso (Orientador)
  Departamento de Estatística, Matemática Aplicada e Computação (DEMAC)
  Universidade Estadual Paulista "Júlio De Mesquita Filho" (UNESP) – Câmpus de Rio Claro
- Prof. Dr. Márcio Andrey Teixeira
  Instituto Federal de São Paulo (IFSP)
  Unidade de Catanduva
- Prof. Dr. Kelton Augusto Pontara da Costa
  Faculdade de Tecnologia (FATEC)
  Unidade Bauru

Conceito: Aprovado

Rio Claro – SP

03 de fevereiro de 2022

Este trabalho é dedicado a minha esposa Ana Carolina,
que sempre me apoiou durante o desenvolvimento deste trabalho.

Agradecimentos

Agradeço a Deus que permitiu de alguma maneira que eu estivesse vivo e com saúde.

Agradeço a minha esposa, Ana Carolina, pelo companheirismo, apoio, suporte emocional e paciência em todos os momentos do desenvolvimento deste trabalho.

Agradeço aos meus pais, Reinaldo e Maria, por toda a estrutura familiar fornecida, que tornou possível minha formação pessoal.

Agradeço ainda meu orientador Prof. Dr. Frank José Afonso, pela parceria, por acreditar em meu potencial, pelos ensinamentos fornecidos, pela paciência e contribuição para minha formação acadêmica.

Agradeço ao amigo Prof. Dr. Osvaldo Severino Junior, que acreditou em meu potencial e autorizou minha licença integral para dedicação ao mestrado acadêmico.

Agradeço ao Instituto Federal de São Paulo pela oportunidade de me afastar integralmente de minhas atividades profissionais, o que viabilizou dedicação a este trabalho de mestrado.

Agradeço ao Laboratório de Redes de Computadores e Sistemas Distribuídos do Instituto Federal de São Paulo (Campus Catanduva) por ceder recursos computacionais que contribuíram para o desenvolvimento deste trabalho.

Por fim, agradeço aos verdadeiros amigos e familiares que compartilharam comigo os bons momentos da vida.

RESUMO

Ao longo da última década nota-se que as pessoas e diferentes tipos de organizações estão se tornando cada vez mais dependentes de sistemas de software, que podem ser caracterizados como aplicações voltadas para *Web*, orientados a serviços e móveis. Em paralelo, observa-se também que aplicações baseadas em serviços (do inglês, *Service-based Applications* – SApps) representa um cenário exponencial de utilização em dispositivos como *smartphones*, *tablets*, *smart* TVs e híbridos. Nessa direção, as atuais tecnologias de conexão 5G e dispositivos de Internet das Coisas prometem aumentar a velocidade de tráfego, diminuir a latência de conexão e aumentar a presença de dispositivos conectados à Internet. Diante desse contexto, as vulnerabilidades e ameaças de segurança oferecem riscos permanentes para este tipo de aplicação (SApps), pois nota-se o surgimento de novas técnicas maliciosas em proporções similares ou superiores. A área de cibersegurança voltada para a camada de aplicação é deficiente em profissionais especializados, além de não existir uma cultura estabelecida entre os desenvolvedores de software em criar aplicações seguras. O cenário exposto mostra que existe uma lacuna entre a dependência da atuação de seres humanos para operacionalizar mecanismos defensivos das aplicações capazes de lidar com ameaças/vulnerabilidades que surgem a todo momento. Motivado pelo cenário exposto, foi conduzido neste trabalho de mestrado acadêmico uma revisão da literatura sobre autoproteção de sistemas baseada em modelos inteligentes, que permite apoiar a detecção de ameaças/vulnerabilidades e ações maliciosas. Com base nos resultados desse mapeamento, uma abordagem de autoproteção para SApps foi desenvolvida para apoiar a detecção de ações maliciosas e a aplicação de contramedidas em tempo de execução. Por meio de *loops* de controle MAPE-K em uma estratégia escalar, modelos de classificação inteligentes podem ser gerados para detectar diferentes tipos de ameaças/vulnerabilidades. Assim, este trabalho apresenta como contribuições principais: (i) o desenvolvimento de processos e soluções que viabilizem a autoproteção de SApps; (ii) um conjunto de boas práticas que podem ser aplicadas durante o projeto e codificação de softwares autoprotegidos; e (iii) um *design* escalável quanto ao número de ameaças/vulnerabilidades que essa abordagem possa implementar. Como forma de avaliar a abordagem proposta neste trabalho, um estudo de caso foi conduzido utilizando uma aplicação chamada `AppPetShop`, uma SApp que foi executada em uma plataforma real de nuvem para simular situações reais de ataques por meio de ferramentas de teste de penetração. Ao final, a abordagem proposta neste trabalho foi comparada com as soluções de autoproteção disponíveis na literatura e os resultados mostram que a mesma pode ser uma solução factível para proteção da camada de aplicação em tempo de execução.

Palavras-chave: Autoproteção; Sistemas autoadaptativos; Ciência de dados; Cibersegurança; Modelos de classificação.

ABSTRACT

Over the last decade, people and different organizations are becoming increasingly dependent on software systems, which can be characterized as web-oriented, service-oriented, and mobile applications. In parallel, we can also observe that Service-based Applications (SApps) represent an exponential scenario of use in devices such as smartphones, tablets, smart TVs, and hybrids. Therefore, current 5G connection technologies and IoT (Internet of Things) devices promise to increase traffic speed, decrease connection latency, and increase devices connected to the Internet. Thus, vulnerabilities and security threats offer permanent risks for these applications, since new malicious techniques are emerging in similar or wider proportions. The cybersecurity focused on the application layer is deficient in expert professionals, and there is no established culture among software developers to create secure applications. This scenario shows that there is a gap between the dependence on performing human beings to operationalize defensive mechanisms in applications and the threats/vulnerabilities that emerge at all times. Motivated by the exposed scenario, this dissertation carried out a review of the literature on self-protecting systems based on intelligent models, which enables to support the detection of threats/vulnerabilities and malicious actions. Based on the results of this review, we developed a self-protection approach for SApps to support the detection of malicious actions at runtime. Basically, through MAPE-K control loops in a scalar strategy, intelligent classification models can be generated to detect different threats/vulnerabilities. Thus, this work presents as main contributions: (i) the development of processes and solutions that enable the self-protection of SApps; (ii) guidelines that can be applied during the design and coding of self-protecting software; and (iii) a scalable design in terms of the number of threats/vulnerabilities this approach can implement. In order to evaluate the approach proposed in this work, a case study was conducted using an application named `AppPetShop`, which can be characterized as a SApp that was executed on a real cloud platform, enabling real situations of attacks to be simulated through penetration testing tools. Finally, we compared the approach proposed in this work with the self-protection solutions available in the literature and the results show it can be a feasible solution to protect the application layer at runtime.

Keywords: Self-protection; Self-adaptive systems; Data science; Cybersecurity; Classification models.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ACL	*Access Control List*
AOP	*Aspect-Oriented Programming*
API	*Application Programming Interface*
AppSec	*Application Security*
ARFF	*Attribute-Relation File Format*
ATM	*Automated Teller Machine*
AUC	*Area Under the ROC Curve*
CBP	*Component-Based Programming*
CI	Critério de Inclusão
CE	Critério de Exclusão
CRC	*Cyclic Redundancy Check*
CRISP-DM	*CRoss Industry Standard Process for Data Mining*
CSV	*Comma-Separated Values*
DDoS	*Distributed Denial of Service*
DoS	*Deny of Service*
ES	Estudo Secundário
Fintech	*Financial technology*
FMDS	*Foundational Methodology for Data Science*
HTTP	*Hypertext Transfer Protocol*
HTTPS	*Hyper Text Transfer Protocol Secure*
IA	Inteligência Artificial
IoT	*Internet of Things*
SaS	*Self-adaptive Systems*
KDD	*Knowledge Discovery in Databases*
LDAPi	*LDAP injection*
MAPE	*Monitor, Analyze, Plan and Execute*
MAPEK	*Monitor, Analyze, Plan and Execute over Knowledge*
MAS	*Multi-Agent Systems*
MVC	*Model-View-Controller*
OSI	*Open Systems Interconnection model*
OTP	*One-Time Password*
OWASP	*Open Web Application Security Project*
PDC	Porcentagem de Detecção Correta
PM	Ponto de Monitoramento
QoS	*Quality of Service*
QP	*Questões de Pesquisa*
RASP	*Runtime Application Self-Protection*
REST	*REpresentational State Transfer*
ROC	*Receiver Operating Characteristic*
SApps	*Service-based Applications*
SGBD	Sistema Gerenciador de Banco de Dados

SOC	*Service-Oriented Computing*
SMS	*Systematic Mapping Study*
SOA	*Service-Oriented Architecture*
SQLi	*SQL injection*
TDSP	*Team Data Science Process*
URI	*Uniform Resource Identifier*
URL	*Uniform Resource Locator*
Weka	*Weikato environment for knowledge analysis*
XSS	*Cross-Site Scripting*

SUMÁRIO

1 INTRODUÇÃO

1.1 Contextualização

A computação orientada a serviços (do inglês, *Service-Oriented Computing* – SOC) aliada a computação em nuvem (do inglês, *Cloud Computing*) tem viabilizado a integração entre dispositivos e software, tornando possível o surgimento de novas plataformas e serviços como, por exemplo, aplicações móveis corporativas e educacionais, cidades e residências inteligentes, monitoramento de saúde em tempo real e transporte (CLEMENT; MCKEE; XU, 2017; CALVERT; KHOSHGOFTAAR, 2019; BOUDKO; ABIE, 2019; LA; KIM, 2018; KACI; RACHEDI, 2019; CALVERT; KHOSHGOFTAAR, 2019). Portanto, pode-se dizer que essas aplicações trouxeram mudanças significativas para pessoas e organizações ao longo dos últimos 10 anos, onde pode-se observar que empregos e organizações foram extintos, remanejados e/ou criados nesse período. Por exemplo, os serviços de *streaming* modificaram o modo como as pessoas consomem conteúdo audiovisual, os serviços de entrega se tornaram mais dinâmicos e acessíveis, o mercado de transporte por aplicativo tem criado novos postos de trabalho, entre outras mudanças (VELDHOVEN; VANTHIENEN, 2019; FIORE *et al.*, 2019). Diante desse cenário, as pessoas e organizações tornaram-se mais dependentes dessas tecnologias, onde a alta disponibilidade e a forte integração com dispositivos móveis passam a sensação de serem onipresentes.

Além do contexto exposto, o surgimento de tecnologias recentes de comunicação como Internet 5G e Internet das coisas (do inglês, *Internet of Things* - IoT) têm impulsionado o aumento do consumo de aplicações *desktop*, móveis e *Web* que implementam comunicação via serviços (SARICA; ANGIN, 2020). Essas aplicações serão referenciadas deste ponto em diante como SApps (do inglês, *Service-based Applications*). Nessa direção, Nord, Koohang e Paliszkiewicz (2019) comentam que em 2025 o número de dispositivos IoT poderá chegar a 75 bilhões. Em paralelo, nota-se que houve um crescimento no número de golpes que podem comprometer ou até mesmo inviabilizar o uso dessas aplicações (SApps) (ZHOU *et al.*, 2020). Em síntese, os criminosos do universo virtual, que podem ser internos à própria organização responsável pelo sistema como também externos a mesma, exploram vulnerabilidades que sequestram navegadores, roubam informações e identidades, tornam serviços indisponíveis, favorecem a pirataria, entre outros golpes (SCHNEIER, 2015). Portanto, pode-se dizer que o crescente aumento dessas tecnologias tem evidenciado a necessidade de viabilizar segurança e/ou proteção à sistemas e dados contra *hackers*.

De acordo com Yang, Zhou e Cui (2020), a implementação de segurança cibernética

em sistemas e dados não depende apenas das tecnologias de defesa, mas também de outras que permitam detectar e descobrir invasões, ameaças e ataques cibernéticos durante a execução de uma aplicação. Nesse sentido, as principais técnicas adotadas para a detecção de ações maliciosas são baseadas em regras e modelos inteligentes. Os métodos baseados em regras podem apresentar baixa taxa de falsos positivos, além de se mostrar um sistema caro e inviável pela dependência que possui do administrador da base de regras, que precisa atualizá-la constantemente para permitir que ocorra a detecção de novos ataques (YANG; ZHOU; CUI, 2020). Em paralelo, na visão de Kaci e Rachedi (2019), os modelos inteligentes visam fazer com que computadores e softwares aprendam a se programar e melhorar com experiência ou dados, permitindo-lhes detectar novos tipos de ações maliciosas por meio da generalização dos modelos inteligentes. Nesse contexto, os algoritmos de aprendizado de máquina se enquadram em uma das seguintes categorias: aprendizado supervisionado, aprendizado não supervisionado e aprendizado por reforço (NORVIG; RUSSELL, 2021). Além de identificar ações maliciosas, um sistema de defesa deve ser capaz de aplicar contramedidas em tempo de execução para que o sistema que está sendo defendido possa continuar operando com qualidade e sem pausas. Nesse contexto, os conceitos sobre sistemas autoadaptativos[1] podem ser um caminho para alcançar autoproteção, pois permite a redução da intervenção humana em contramedidas aos ataques. Assim como o sistema nervoso humano que controla automaticamente os ajustes do corpo, como o sistema digestivo, excretor, respiratório, entre outros, um mecanismo de proteção pode monitorar um software, detectar ações maliciosas, e modificar seu comportamento e/ou estrutura em resposta a algum tipo de mudança que ocorre no próprio sistema, em seu ambiente ou mesmo em seus objetivos (HUEBSCHER; MCCANN, 2008; MONTRIEUX; LEMOS; BAILEY, 2019).

1.2 Motivação

Diante do contexto exposto e motivado em entender o tema de pesquisa em torno das áreas envolvidas, uma revisão da literatura foi conduzida durante o desenvolvimento deste trabalho de mestrado. O principal objetivo dessa revisão foi estabelecer um panorama sobre iniciativas de autoproteção que utilizem técnicas inteligentes para detecção de ações maliciosas em SApps. A técnica de mapeamento sistemático (do inglês, *Systematic Mapping Study* – SMS) foi utilizada nessa revisão (KERN; KESAVAN; DASWANI, 2007; KITCHENHAM; DYBA; JORGENSEN, 2004; PETERSEN *et al.*, 2008; PETERSEN; VAKKALANKA; KUZNIARZ, 2015), pois permite que seja possível conduzir uma avaliação completa e justa sobre um determinado tema por meio de uma metodologia confiável e isenta de influências pessoais.

O número de estudos reduzido do referido mapeamento evidencia que a temática

[1] O termo sistemas autoadaptativos tem sido utilizado em diferentes áreas/domínios. Como nesta dissertação, esse termo se concentrará apenas no domínio de software, o mesmo será referenciado deste ponto em diante como software autoadaptativo (do inglês, *Self-adaptive Software* – SaS)

investigada neste trabalho é uma área pouco explorada pela academia. Nesse sentido, buscou-se entender os seguintes aspectos: (i) **engenharia de sistemas**, onde buscou-se entender quais abordagens de *design* e arquiteturas são utilizadas para prover a autoproteção. Sobre esse aspecto verificou-se que um número significativo de trabalhos teve como proposta principal apresentar melhorias nas técnicas e processos relacionados a criação de modelos inteligentes, sendo pouco exploradas as questões relacionadas à engenharia dessas propostas (LA; KIM, 2018; CALVERT; KHOSHGOFTAAR, 2019; YANG; ZHOU; CUI, 2020; DENG *et al.*, 2019; TUN *et al.*, 2018; KHANJI; KHATTAK, 2020); (ii) **camadas monitoradas**, onde buscou-se entender em quais camadas da aplicação ocorre o monitoramento em busca de ameaças/vulnerabilidades. Em relação ao conjunto de trabalhos do mapeamento, um número relevante de estudos identificados explora o monitoramento das camadas periféricas da aplicação, o que dificulta a identificação de ações maliciosas que exploram falhas de *design* e/ou validação de dados (BENZAID; TALEB, 2020; BOUDKO; ABIE, 2019; Tariq *et al.*, 2020; ALLEN; GOLOUBEW, 2020; SARICA; ANGIN, 2020; KACI; RACHEDI, 2019; RESTUCCIA *et al.*, 2019); e (iii) **capacidade de resiliência**, onde sistemas devem entregar os resultados esperados e se recuperar após sofrerem ciberataques. Sobre esse aspecto foram avaliadas as capacidades consideradas essenciais para garantir resiliência, que representa uma característica composta por três atividades: (i) capacidade de uma solução em detectar de maneira proativa e reativa as ações maliciosas; (ii) modificação em tempo de execução em contramedida às ações maliciosas detectadas; e (iii) especificar claramente quais dimensões de segurança devem ser protegidas.

Analisando o rol de trabalhos do mapeamento conduzido, nenhum foi capaz de atender aos três aspectos supracitados. Em paralelo, pode-se afirmar que os trabalhos avaliados não elencam iniciativas claras relacionadas à boas práticas de segurança durante o processo de desenvolvimento das aplicações. Nesse sentido, de acordo com OWASP (2021d), boa parte das vulnerabilidades de segurança são causadas por falhas provenientes do processo de desenvolvimento, podendo ser relacionadas ao *design* ou codificação de uma aplicação.

1.3 Objetivos

Este trabalho tem como objetivo geral estabelecer uma abordagem que seja capaz de viabilizar autoproteção em SApps voltada para a camada de aplicação. Em linhas gerais, a abordagem deve ter foco em três pilares, a saber: (i) definição de arquitetura e *design* que sejam capazes de prover autoproteção em sistemas de modo que permitam ações proativas e reativas de segurança, sendo suficientemente flexíveis para mitigar as ameaças/vulnerabilidades atuais e futuras; (ii) utilizar modelos inteligentes para apoiar de detecção de ameaças/vulnerabilidades; (iii) estabelecer boas práticas que devam ser conduzidas durante o processo de desenvolvimento dos softwares protegidos. Para que esse objetivo geral seja alcançado, os seguintes Objetivos

Eespecíficos (OE) são propostos:

OE-1. Estabelecer diretrizes e processos de segurança da informação que guiem a fase de projeto e desenvolvimento de SApps, onde seja possível estabelecer responsabilidades para as atividades de codificação, ciência de dados e segurança da aplicação protegida.

OE-2. Definir um modelo de *design* que permita prover autoproteção em SApps de modo não intrusivo e que seja escalável o suficiente para suportar futuras ameaças/vulnerabilidades.

OE-3. Estabelecer uma integração entre os processos de desenvolvimento de software, ciência de dados e monitoramento de segurança.

OE-4. Implementar soluções que permitam automatizar os processos que compõem a abordagem de autoproteção, a saber são ciência de dados, desenvolvimento e cibersegurança.

OE-5. Avaliar e validar a abordagem de autoproteção proposta neste trabalho por meio de um estudo que configure um cenário computacional real.

OE-6. Apresentar um panorama detalhado referente ao estágio atual, desafios de pesquisa e perspectivas futuras sobre abordagens de autoproteção para SApps por meio de um mapeamento sistemático.

1.4 Organização da dissertação

O Capítulo 2 introduz a fundamentação teórica necessária para compreender o conteúdo dos capítulos subsequentes desta dissertação. Neste sentido, são abordadas definições e conceitos sobre sistemas autoadaptativos, segurança e autoproteção em aplicações SOC, Web e móveis, e ciência de dados voltado à cibersegurança. No Capítulo 3 é apresentado a condução do mapeamento sistemático da literatura, onde buscou-se compreender o estado da arte em relação à autoproteção de sistemas com auxílio de modelos inteligentes. Esse capítulo inicia com uma investigação preliminar sobre estudos secundários relacionados ao tema deste trabalho de mestrado, na sequência é apresentado o mapeamento da literatura e, por fim, são reportadas as considerações finais. O Capítulo 4 apresenta a abordagem de autoproteção proposta nesta dissertação, sendo seus processos organizados em quatro atividades. Em seguida, são apresentadas recomendações e boas práticas que podem ser conduzidas para a implementação, implantação e manutenção da aplicação protetora. Ao final do capítulo são apresentadas as considerações finais a respeito da abordagem apresentada. No Capítulo 5 é apresentado uma visão detalhada do estudo de caso conduzido para avaliar e viabilidade da abordagem de autoproteção proposta. Por fim, o Capítulo 6 apresenta as contribuições oriundas desta dissertação, bem como perspectivas para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são apresentados conceitos e definições dos temas envolvidos neste trabalho. Inicialmente, Seção 2.1 é apresentada uma visão geral sobre SaS, as propriedades *self-** e as principais abordagens de engenharia para esse tipo de sistema de software. Na Seção 2.2 são reportadas as principais definições e conceitos sobre segurança de sistemas e autoproteção/segurança adaptativa. Na Seção 2.3 são apresentados os principais conceitos sobre ciência de dados, suas abordagens de desenvolvimento, e aplicabilidade na área de cibersegurança. Por fim, na Seção 2.4 são apresentadas as considerações finais sobre o conteúdo apresentado neste capítulo.

2.1 Sistemas autoadaptativos

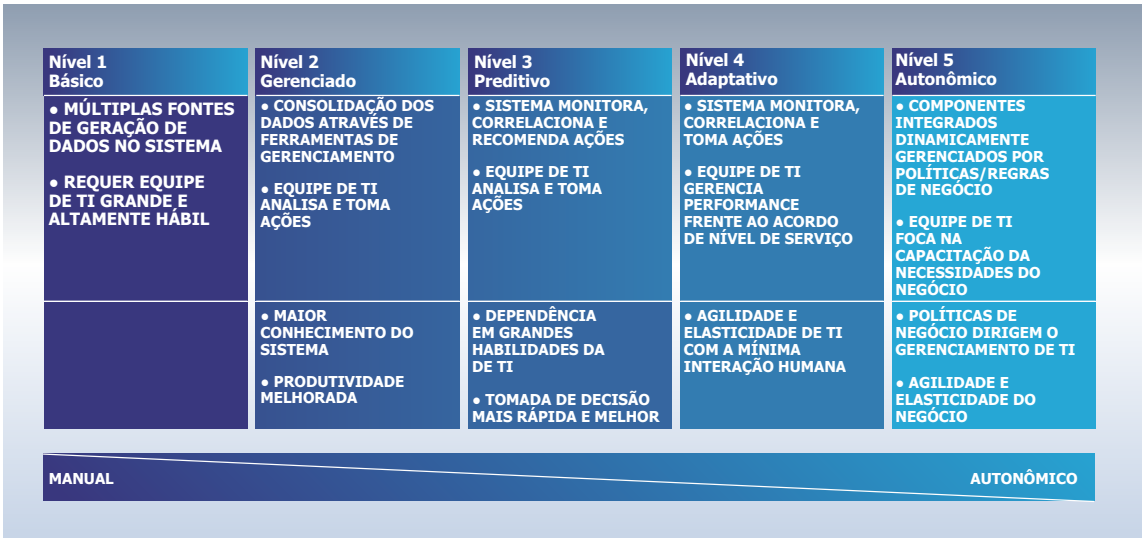
De acordo com Salehie e Tahvildari (2009), SaS pode ser definido como uma classe especial de sistemas de software por lidar como modificação de sua estrutura e/ou comportamento em tempo de execução. Para isso, esse tipo de sistema utiliza uma estrutura de controle conhecida como MAPE-K (do inglês, *Monitor, Analyze, Plan, Execute over Knowledge*) (IBM, 2005). Além disso, o modelo 5W1H também tem sido utilizado para auxiliar os engenheiros de software na identificação dos requisitos e, consequentemente, no projeto desse tipo de software (SALEHIE; TAHVILDARI, 2009). Como forma de apresentar esses assuntos, esta seção foi organizada da seguinte maneira: na Seção 2.1.1 são apresentados conceitos e definições sobre SaS; na Seção 2.1.2 é apresentado um panorama sobre as principais propriedades (do inglês, *self-* properties*) e o modelo 5W1H; na Seção 2.1.3 são reportados os conceitos sobre a aplicabilidade da engenharia de software em SaS; e, por fim, na Seção 2.1.4 são apresentados abordagens para desenvolvimento de SaS.

2.1.1 Conceitos e definições sobre SaS

A computação autônoma é um paradigma inspirado em sistemas biológicos que visa lidar com a administração de sistemas computacionais complexos, oferecendo possibilidades de autogerenciamento (do inglês, *self-management*) sem a necessidade de intervenção humana (KHALID *et al.*, 2009). Nesse sentido, visando classificar a maturidade desse tipo de sistema, Kephart e Chess (2003) elaboraram um modelo organizado em cinco níveis, como ilustra a Figura 1. O Nível 1 (**Básico**) possui muitos processos que são executados manualmente pela equipe de tecnologia da informação, se comparado com o nível mais automatizado (Nível 5 – **Autonômico**). O Nível 4 (**Adaptativo**) difere do Nível 5, pois o sistema se modifica de acordo com as mudanças

ocorridas no ambiente. No Nível 5, as modificações são feitas de acordo com as mudanças e políticas de negócio implantadas no sistema. Nos níveis 2 (**Gerenciado**) e 3 (**Preditivo**), a equipe de TI analisa os dados expostos pelo sistema e toma alguma ação, que tem como referência as recomendações apresentadas/sugeridas pelo nível 3.

Figura 1 – Níveis de maturidade da computação autônoma.



Fonte: Adaptado de (BRAGA *et al.*, 2006)

De acordo com Krupitzer *et al.* (2015), a complexidade dos sistemas de informação tem aumentado de maneira significativa nos últimos anos, levando a um maior esforço de manutenção e aumento nos custos pós-implantação. A incorporação de características de autoadaptação em tais sistemas tem se mostrado como uma alternativa viável para contornar essas adversidades. Para alcançar o comportamento adaptativo, as propriedades básicas do sistema são: autoconsciência (do inglês, *self-awareness*) e contextualização (do inglês, *context-awareness*). A primeira descreve a capacidade de um sistema estar consciente de si mesmo (ou seja, requisitos funcionais), sendo capaz de monitorar seus recursos, estado interno e comportamento. A segunda significa que o sistema está ciente de seu ambiente operacional (ou seja, não funcionais) em que está executando, que geralmente lida com as estruturas externas do sistema.

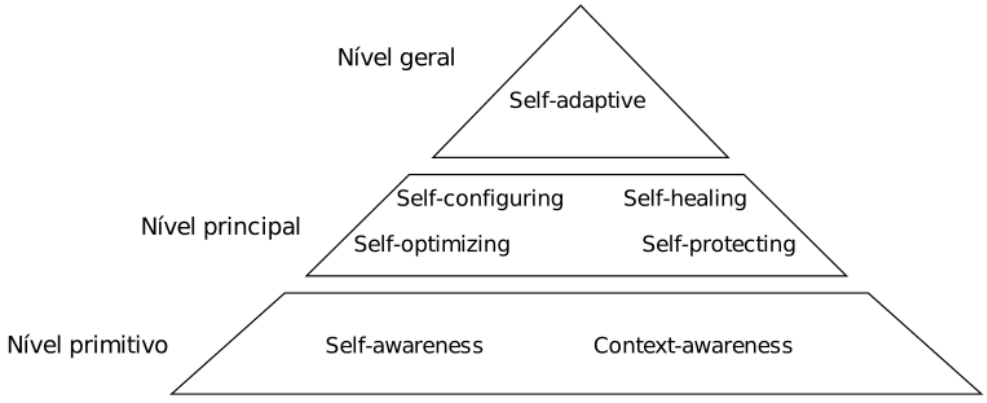
Para Salehie e Tahvildari (2009), as adaptações de um sistema podem ocorrer em decorrência de causas internas e/ou externas, que podem envolver, por exemplo, eventos externos, aumento de solicitações dos usuários, ou situações não previstas na fase de *design* do software. Para monitorar as causas supracitadas, o SaS deve implementar um mecanismo de adaptação para monitorar o próprio sistema e seu contexto para detectar mudanças e decidir como (re)agir em função de tais mudanças. O autor comenta ainda que o SaS é um sistema de circuito fechado com *feedback* do sistema e seu contexto. Portanto, espera-se que um mecanismo de adaptação tenha a capacidade de rastrear as alterações de software e tomar as ações apropriadas em tempo

de execução, com o mínimo de intervenção humana. O ponto chave para o SaS é que seu ciclo de execução não deve ser interrompido após o desenvolvimento e configuração inicial. Esse ciclo deve ser mantido após a instalação, a fim de avaliar o sistema e responder às mudanças ao longo do tempo. Na visão desses autores, o SaS deve alterar seu comportamento ou estrutura sempre que o contexto sofrer alterações. Portanto, sempre que o sistema decide se adaptar, isso pode levar à necessidade de atividades de verificação para fornecer avaliação contínua, que visa preservar a qualidade do software. Para isso, o SaS avalia sua própria situação em relação aos objetivos iniciais de maneira contínua por meio de diferentes estratégias de adaptação. Por fim, é fundamental que o próprio sistema possa garantir que o comportamento desejado não seja comprometido durante o processo de adaptação.

2.1.2 Panorama sobre propriedades self-*

Segundo Khalid *et al.* (2009), a computação autônoma gira em torno de oito propriedades *self-** definidas pela IBM. A Figura 2 é apresentada a hierarquia de propriedades *self-** organizada em três níveis, os quais são descritos a seguir.

Figura 2 – Hierarquia de propriedades *self-**



Fonte:Adaptado de (SALEHIE; TAHVILDARI, 2009)

No topo da hierarquia (Nível Geral) encontra-se a propriedade autoadaptação (do inglês, *Self-adaptive*), que normalmente consiste em muitos elementos desconhecidos ou com apenas um conhecimento parcial sobre o sistema global. No nível principal, foram definidas quatro propriedades *self-** baseadas em mecanismos de autoadaptação biológica (KEPHART; CHESS, 2003), são elas:

Autoconfiguração (do inglês, *Self-configuring*): é a capacidade de reconfigurar automaticamente seguindo políticas pré-definidas em alto nível, facilitando a adaptação em relação às mudanças causadas por configurações automáticas;

Auto-otimização (do inglês, *Self-otimizing*): é a capacidade de automonitorar continuamente e controlar recursos para melhorar o desempenho e eficiência do software, visando satisfazer os requisitos de diferentes usuários;

Autocura (do inglês, *Self-healing*): é a capacidade de detectar, diagnosticar e reparar falhas automaticamente. Essa propriedade pode antecipar possíveis problemas e, consequentemente, tomar as ações adequadas para evitar uma falha; e

Autoproteção (do inglês, *Self-protecting*): é a capacidade de identificar proativamente violações de segurança ou proteger a si mesmo de ataques maliciosos ou falhas em cascata, que não são corrigidas pelas medidas tomadas na autocura.

Na base da pirâmide proposta (Nível primitivo – Figura 2), estão as propriedades primitivas autoconsciência (do inglês, *Self-awareness*) e consciência de contexto (do inglês, *Context-awareness*). A primeira indica que o sistema está ciente de seus próprios estados e comportamentos. Já a segunda sinaliza que o sistema está ciente de seu contexto, que representa seu ambiente operacional.

De acordo com Salehie e Tahvildari (2009), independentemente do nível de hierarquia adotado em um SaS, o mesmo deve utilizar alguma prática de engenharia de requisitos para identificação dos interesses do sistema, tais como: escopo, restrições, nível de adaptação, entre outros. Os autores mencionam o modelo 5W1H como uma alternativa viável para elicitação de requisitos no contexto de SaS. Em síntese, esse acrônimo representa seis questões que devem ser respondidas para que se possa obter os recursos essenciais para o desenvolvimento desse tipo de software. A seguir, uma descrição de cada questão é apresentada:

Onde (do inglês, *Where*): esta questão visa responder em que local está a necessidade de mudança, ou seja, quais camadas ou artefatos dos *software* sofrerá a adaptação.

Quando (do inglês, *When*): esta questão visa identificar os interesses temporais sobre o processo de adaptação. Quando uma mudança precisa ser aplicada e quando é possível fazê-la? Com que frequência o sistema precisa ser alterado? Essas mudanças são reativas ou proativas?

O que (do inglês, *What*): esta questão visa identificar quais atributos ou artefatos do sistema podem ser alterados e o que precisa ser alterado em cada cenário de adaptação. Em resumo, as modificações podem ocorrer em parâmetros, métodos, componentes, estilo de arquitetura, e recursos do sistema.

Por que (do inglês, *Why*): esta questão visa identificar as motivações da construção de SaS, referindo-se aos objetivos abordados pelo sistema, como, por exemplo, robustez, escalabilidade, entre outros.

Quem (do inglês, *Who*): esta questão aborda o nível de automação e envolvimento humano no processo de adaptação. Espera-se que a intervenção humana seja mínima quando o sistema estiver no ambiente de execução. A interação com proprietários e gerentes de sistema é necessária para criar políticas de confiança e transferência na fase de *design*.

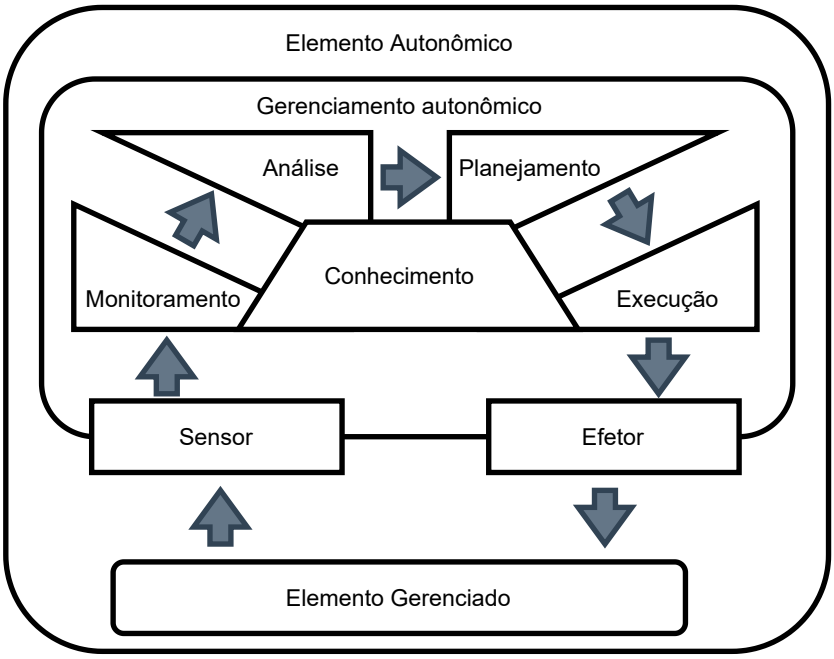
Como (do inglês, *How*): esta questão determina como os artefatos adaptáveis podem ser alterados e quais ações de adaptação podem ser apropriadas para serem aplicadas em uma determinada condição.

Salehie e Tahvildari (2009) recomendam que as questões do modelo 5W1H sejam respondidas em duas fases: projeto e tempo de execução. Na primeira, os projetistas extraem requisitos com base nas perguntas e estabelecem mecanismos e alternativas a serem usados na fase operacional. Já na segunda, o sistema é adaptado com base nas questões levantadas na fase de projeto, nas questões respondidas por administradores e gerentes por meio de políticas, nas análises do próprio sistema. Além disso, vale destacar que o envolvimento humano na fase operacional traz desafios complexos ao SaS, que incluem gerenciamento de políticas, confiança e atividades manuais. No tocante ao gerenciamento de políticas, a representação explícita dos objetos é uma questão em aberto em muitos sistemas. Em relação à construção de confiança por partes dos gerentes e administradores de sistemas, o sistema deve tornar rastreável em cada uma das atividades que ocorrem no processo de autoadaptação.

2.1.3 Engenharia para SaS

Esta seção apresenta uma visão geral sobre engenharia para SaS. Para isso, uma visão geral sobre o laço de controle, elemento essencial para o projeto de SaS, e a organização da lógica de adaptação são reportados. O gerenciador autônomo é responsável pelo controle de recursos gerenciáveis, operando por meio de um laço de controle chamado MAPE-K. Em resumo, esse laço atua em quatro atividades (Monitorar, Analisar, Planejar e Executar) sobre uma base de conhecimento. Essa base armazena informações do atual estado do ambiente computacional e políticas informadas pelos administradores do sistema, compartilhando tais informações entre as quatro atividades do laço de controle MAPE (do inglês, *Monitor, Analyze, Plan, Execute*) (KEPHART; CHESS, 2003; IBM, 2005). A comunicação entre o gerenciador autônomo e o recurso gerenciável ocorre por meio de uma interface de gerenciamento (do inglês, *Manageability Interface*). Essa interface é composta por elementos sensores (do inglês, *Sensor*) e efetores (do inglês, *Effector*). O elemento *Sensor* é responsável por coletar informações do recurso gerenciável e passar para o processo de monitoramento do gerenciador autônomo. O elemento efetor tem a função de executar as operações enviadas pela parte de execução do gerenciador autônomo no recurso gerenciável (KEPHART; CHESS, 2003). A Figura 3 ilustra a estrutura do *loop* MAPE-K, onde as setas representam o sentido do fluxo das informações.

Figura 3 – Estrutura geral do laço de controle.



Fonte: (KEPHART; CHESS, 2003; IBM, 2005)

De acordo com Cheng *et al.* (2009), existem alguns detalhes importantes de engenharia que não são explicitados pelo modelo MAPE-K. O primeiro é voltado para coleta de dados, pois é nesta fase em que os sensores coletam informações do sistema em seu ambiente de execução, refletindo o atual estado do sistema. Para isso, é necessário calibrar uma taxa de amostragem adequada dos dados, além de definir um formato comum de eventos entre os sensores. A segunda refere-se a análise dos dados, sendo esta fase responsável por estruturar e raciocinar sobre os dados brutos coletados na fase anterior. Portanto, é importante planejar a quantidade de dados históricos que serão armazenados para permitir que o sistema possa raciocinar sobre as futuras adaptações e sobre o processo de validação e verificação pelos administradores do sistema. Por fim, a decisão de adaptação representa o processo que atuará na modificação do sistema por meio dos atuadores e efetores disponíveis no *loop*. Em sistemas onde são implementados vários circuitos de adaptação é necessário garantir que a interferência entre as propriedades implementadas não seja prejudicial para o sistema.

De acordo com Lemos *et al.* (2013), o gerenciador de recursos de um sistema autoadaptativo pode ser classificado em duas categorias: (i) centralizado ou descentralizado; e (ii) distribuído e não distribuído. Na primeira, os gerenciadores de recursos descentralizados estão centrados no aspecto do algoritmo de adaptação, onde não há um único componente que possua as informações completas do estado do sistema, e os processos tomam decisões de adaptação baseadas apenas nas informações locais. Em um sistema autoadaptativo centralizado, por outro lado, as decisões sobre as adaptações são tomadas por um único componente. Já na segunda, os gerenciadores de

recursos descentralizados consistem em vários componentes de software implantados em vários *hosts* conectados por algum tipo de rede como, por exemplo, em um ambiente de nuvem. A maneira em que ciclos de controle de gerenciador de recursos orquestram o processo de adaptação podem variar em cinco tipos, a saber (LEMOS *et al.*, 2013):

Orquestração Hierárquica. O controle feito pelo gerenciador de recursos ocorre de maneira hierárquica, onde cada camada possui uma estrutura MAPE completa. O objetivo de cada uma das hierarquias geralmente está relacionado a uma escala de tempo, onde as camadas inferiores possuem escalas menores em relação às superiores. Em resumo, essas camadas podem compartilhar informações de controle com as demais, permitindo então uma sincronização de execução das estruturas MAPE.

Orquestração Mestre e Escravo. Os componentes de análise e planejamento da estrutura MAPE exercem uma relação de mestre-escravo entre os componentes de monitoramento e execução. Em síntese, esses componentes coletam informações e enviam para os componentes de análise e planejamento que estão dispostos de maneira centralizada.

Orquestração Regional. Diversos *hosts* em uma camada inferior, que contém em sua estrutura os componentes de monitoramento, análise e execução do modelo MAPE, estão relacionados hierarquicamente a um *host* regional e centralizado em uma camada superior, a qual possui o componente de planejamento. Os *hosts* da camada inferior coletam os dados do sistema e seu ambiente operacional e enviam para o componente de análise, que ao finalizar a análise dos dados brutos, encaminham os resultados para o *host* regional. O componente de planejamento do *host* regional obtém as análises da sua região e, com um conhecimento global do sistema, cria um plano de execução e envia para todos os *hosts* de nível inferior para efetuar as execuções.

Orquestração totalmente descentralizada. Neste tipo, cada *host* implementa uma estrutura MAPE completa, cujos componentes locais coordenam sua operação com os componentes pares correspondentes de outros *hosts*. Portanto, o compartilhamento de informações entre os *hosts* permite que ações de adaptação sejam coordenadas.

Orquestração que compartilha informações. Este tipo de orquestração é semelhante ao descentralizado, onde cada *host* possui uma estrutura completa MAPE diferenciando-se pelo fato de apenas o componente de monitoramento compartilhar dados com outros *hosts*. Portanto, as informações coletadas sobre o *status* dos sistemas gerenciados são compartilhadas entre os vários monitores enquanto a análise dos dados coletados é realizada. A decisão sobre possíveis ações de adaptação são executadas sem nenhuma ação de coordenação com os outros *hosts*.

2.1.4 Abordagens para desenvolvimento de SaS

Em Krupitzer *et al.* (2015), um estudo sobre abordagens para engenharia de SaS foi desenvolvido. Tal estudo mostra como essas abordagens atuam em diferentes segmentos de desenvolvimento e como a lógica de adaptação deve ser organizada. De acordo com os autores, essas abordagens foram organizadas em seis tópicos, a saber: arquitetura, reflexão, centradas em paradigmas de programação, serviços, agentes e aprendizado. A seguir, detalhes de cada abordagem são apresentados.

Abordagem baseada em arquitetura. Em geral, uma arquitetura de software é organizada em um conjunto de camadas, componentes e/ou conectores. Um SaS deve ter autoconsciência para que possa estar ciente de sua arquitetura e fazer uso desse conhecimento para raciocinar sobre os níveis e processos de adaptação. Em síntese, os modelos de arquitetura são usados nesse tipo de abordagem em combinação com métricas como políticas, metas, estratégias, e/ou restrições de adaptação para que o software tenha a capacidade de raciocinar sobre as mudanças a serem realizadas. Além disso, os sistemas que organizam a lógica de adaptação em uma camada externa em relação ao software adaptável têm obtido melhores resultados, pois esta lógica fica organizada como componentes dedicados que podem ser reutilizados em vários sistemas, além de terem sua complexidade reduzida pela segmentação de código (ou seja, lógica de adaptação e software adaptável).

Abordagem baseada em reflexão. A reflexão está relacionada à propriedade adaptativa de autoconsciência, sendo definida pela capacidade do software de examinar e, possivelmente, modificar sua estrutura e/ou comportamento em tempo de execução. A reflexão pode ser classificada em introspecção, que é a capacidade de observar o comportamento de uma aplicação, e intercessão, que é a capacidade de reagir aos resultados da introspecção. A introspecção pode ser usada na adaptação estrutural, sendo por parâmetro ou contexto. O motivo da adaptação pode ser detectado via monitoramento e análise, que permite apenas a adaptação reativa. Já a intercessão determina e controla a adaptação, o planejamento e a execução, ou seja, os resultados da introspecção. Por esse motivo a intercessão pode ser implementada em situações de adaptação preditiva.

Abordagem baseada em paradigmas de programação. Alguns paradigmas de programação proporcionam a possibilidade de adaptação por parâmetros e estrutura. Os principais paradigmas de programação são: (i) baseado em componentes (do inglês, *Component-Based Programming* – CBP), que representam partes de software encapsuladas que podem ser desenvolvidas, implantadas e compostas de maneira independente; (ii) orientado a aspectos (do inglês, *Aspect-Oriented Programming* – AOP), o software é dividido em interesses de negócio e preocupações transversais. A recomposição dinâmica está frequentemente relacionada às preocupações transversais como qualidade de serviço e balanceamento de carga; e (iii) orientado

a contexto (do inglês, *Context-oriented Programming*), onde o contexto é incorporado como uma construção de primeira classe em linguagens de programação. Nesse sentido, pode-se dizer que o software é capaz de raciocinar sobre mudanças no contexto e se adaptar adequadamente.

Abordagem baseada em serviços. Os serviços são unidades de software pequenas, encapsuladas e autônomas que cumprem uma tarefa específica. Em síntese, os serviços são usados para apoiar o desenvolvimento de aplicativos rápidos, de baixo custo, interoperáveis, evoluíveis e distribuídos em massa. Em sistemas orientados a serviços com características de adaptação, a lógica de adaptação deve nortear a execução dos serviços, provendo modificações em tempo de execução para que a aplicação permaneça em funcionamento.

Abordagem baseada em agentes. Um agente é um software que cumpre uma tarefa específica de forma autônoma, além de cooperar com outros agentes para execução de tarefas complexas. Um sistema multiagente (do inglês, *Multi-Agent Systems* – MAS) é formado por um conjunto de agentes que compartilham objetivos comuns, que se comunicam e cooperam entre si para execução de uma tarefa. É possível observar na literatura que MAS e SOA (do inglês, *Service Oriented Architectures*) têm sido integrados para construir SaS com base em agentes autônomos, além de modelos de objetos adaptáveis para projetar um MAS ou apresentam padrões de *design* para SaS baseados em estruturas de agentes.

Abordagem baseada em aprendizado. O aprendizado em SaS está fortemente associado à auto-otimização. Um SaS otimiza continuamente sua estrutura, parâmetros ou algoritmos para se tornar mais eficiente em relação ao desempenho ou custo, concentrando o aprendizado na otimização estrutural. Em geral, as abordagens baseadas em aprendizado focam principalmente no componente de planejamento, que pode ser usado para a adaptação/evolução da própria lógica de adaptação.

2.2 Autoproteção de sistemas

Esta seção apresenta conceitos, definições e detalhes sobre autoproteção de sistemas, sendo destacada a combinação entre SaS e suas propriedades *self-** com a área de segurança de sistemas. Dentre as propriedades de nível principal, *self-configuring*, *self-otimizing*, *self-healing* e *self-protecting*), esta seção trata especificamente sobre a última do referido nível no contexto de aplicações orientadas a serviços. Como forma de apresentar os assuntos supracitados, esta seção foi organizada da seguinte como segue. A Seção 2.2.1 são apresentados conceitos e definições sobre segurança em linhas gerais para aplicações orientadas a serviços e Web. Na sequência são abordadas as dez ameaças mais recorrentes para sistemas e APIs (do inglês, *Application Programming Interface*) Web, conforme relatórios divulgados pela organização OWASP (do inglês, *Open Web Application Security Project*). Já na Seção 2.2.2 são reportados definições e

conceitos sobre autoproteção e segurança adaptativa, um sinônimo para o assunto desta seção.

2.2.1 Segurança de sistemas

Na visão de Schneier (2015), a segurança não deve ser observada apenas sob a ótica de produtos individuais, mas sim de sistemas e cada um dos componentes que os compõem. De acordo com o autor, as ameaças do mundo digital possuem as mesmas motivações do mundo físico, refletindo os mesmos problemas, tais como: roubo, extorsão, quebra de privacidade, vandalismo, exploração, entre outros. A Internet possui três elementos que tornam as ameaças virtuais mais devastadoras em relação às físicas, a saber: automatização de tarefas, ações maliciosas a distância, e a rápida propagação de técnicas maliciosas pela rede. De acordo com Kern, Kesavan e Daswani (2007), a segurança de sistemas consiste em sete conceitos principais, os quais são descritos a seguir:

Autenticação. É a atividade de verificar a identidade de um usuário ou dispositivo. A identificação de identidade de usuários pode ser baseada em algo que o usuário (i) conheça (por exemplo, uma senha secreta), (ii) possua (por exemplo, cartões OTP (do inglês, *One-Time Password*), cartões inteligentes ou ATM (do inglês, *Automated Teller Machine*)), e (iii) seja (por exemplo, geralmente utilizadas técnicas biométricas como digitais e íris). Em sistemas distribuídos, a comunicação pode ocorrer entre máquinas, que normalmente suportam três tipos de autenticação: autenticação do cliente, onde o servidor verifica a identidade do cliente; autenticação do servidor, onde o cliente comprova a identidade do servidor; e autenticação mútua, onde o cliente e o servidor fazem uma verificação mútua.

Autorização. É a atividade de verificar a autoridade de um usuário ou dispositivo para acessar algum objeto ou realizar alguma ação em um sistema. Para controlar a autoridade de um usuário ou dispositivo, geralmente é utilizado o modelo de ACL (do inglês, *Access Control List*), em que existe um conjunto de usuários e de recursos, além da relação de autoridade entre ambos.

Confidencialidade. Tem como objetivo manter o conteúdo de uma comunicação, armazenamento temporário ou persistente em sigilo. Sistemas distribuídos geralmente utilizam tecnologias de criptografia para manter a confidencialidade dos dados que são transferidos entre os *hosts*.

Integridade de mensagens e dados. Esta atividade é semelhante à confidencialidade, diferenciando-se pelo fato de que um ataque malicioso não apenas visualizará o conteúdo, mas também tentará fazer a alteração dos dados. Para evitar esse tipo de ataque, pode ser utilizado a técnica de CRC (do inglês, *Cyclic Redundancy Check*), que permite alcançar a integridade e detectar quando *bits* em uma mensagem foram perdidos ou alterados.

Responsabilidade. O objetivo é garantir que seja possível determinar o autor de uma ação maliciosa. Geralmente os sistemas computacionais atribuem responsabilidades por meio de autenticação e uso de trilhas de registro, as quais são conhecidas também como "log". Os logs podem ser registrados a partir da autenticação de usuário ou realização de atividades em um sistema.

Disponibilidade. É a capacidade de responder às solicitações em um prazo considerado razoável. Ataques de negação de serviço (do inglês, *Deny of Service* – DoS) são comumente deferidos contra sistemas de informação, onde são enviadas quantidades excessivas de tráfego de rede a um serviço. O objetivo desse tipo de ataque é tornar o serviço sobrecarregado ou incapaz de responder às requisições de usuários legítimos. Os ataques de negação de serviço podem ser deferidos a alvos de maneira distribuída através da técnica DDoS (do inglês, *Distributed Denial of Service*). Em ataques DDoS, criminosos comandam *hosts* remotamente e, de maneira orquestrada, estes *hosts* disparam ataques ao mesmo tempo a um destino. Um dos meios de defesa a esses tipos de ataques é inserir redundâncias no sistema para eliminar pontos de falha.

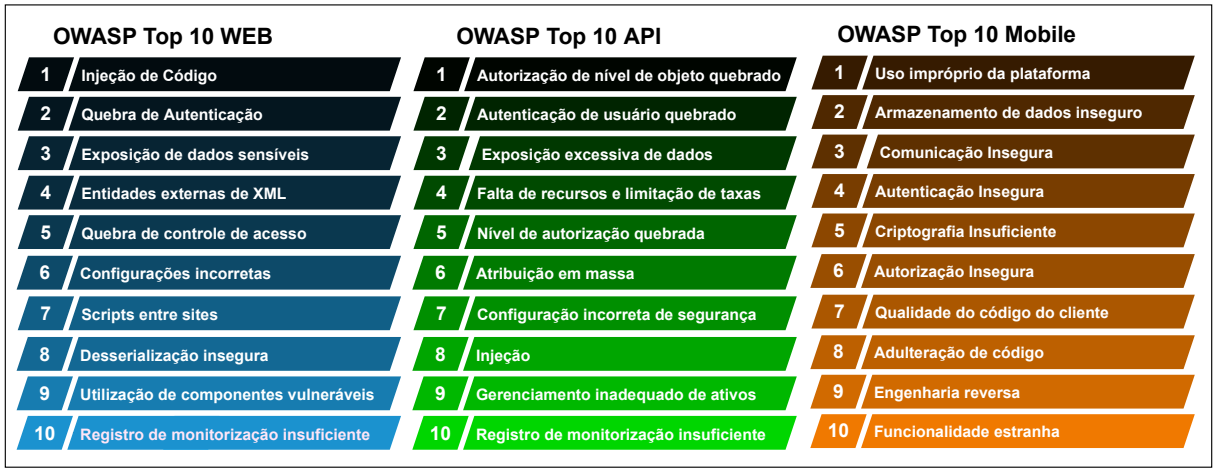
Não repúdio. Em geral, os protocolos de não repúdio são usados para garantir que duas partes não possam negar que interagiram entre si.

Para Cross (2007), os tipos de ameaças/vulnerabilidades existentes em aplicações Web que mais se destacam são: (i) manipulação indevida de campos ocultos que pode revelar informações que não devem ser disponibilizadas; (ii) adulteração de valores de parâmetros enviados a um servidor, que por meio de formulários ou *hyperlinks* são modificados, a fim de levar a resultados inesperados de processamento por parte do servidor; (iii) inserção de trechos de programas interpretáveis, conhecidos como *scripts*, nos dados que são trafegados entre cliente e servidor. Em geral, esses *scripts* são disfarçados como dados legítimos e são invocados no lado clientes para execução de tarefas maliciosas; (iv) estouro de *buffer* ou excesso de dados pode sobrecarregar o software e causar funcionamentos indesejados; e, por fim, (v) envenenamento por *cookies*, onde usuários mal-intencionados podem alterar os dados dos *cookies* que são armazenados em um computador e, caso o servidor não realize validações satisfatórias, execuções maliciosas podem ocorrer. Ainda nessa direção, o relatório das 25 maiores ameaças de softwares produzido pela CWE (2019), aponta que muitas das falhas de segurança de aplicações Web são causadas por erros humanos durante a fase de desenvolvimento. Em linhas gerais, os códigos das aplicações necessitam de revisões de segurança para que falhas e/ou vulnerabilidades sejam mitigadas. Culturalmente, os desenvolvedores de software não são ensinados a lidar com essas fraquezas em suas formações e a maioria não recebe treinamento no trabalho sobre esse assunto. Portanto, recomenda-se que o código fonte de aplicativos seja verificado para constatar se os controles lógicos e de segurança estão presentes, se os mesmos estão funcionando como planejado e se

foram invocados nos devidos locais (OWASP, 2017a). De acordo com Malek *et al.* (2019), a revisão manual de código de segurança não é considerada a melhor prática para softwares de grande escala que consistem em milhões de linhas de código. Na visão dos autores, avaliar manualmente o código não é apenas trabalhoso, mas também é intrinsecamente propenso a erros e pode resultar em problemas de segurança não identificados.

O OWASP[1] é uma organização que visa fomentar o desenvolvimento de segurança voltado a vários domínios de aplicações. Na Figura 4 são ilustrados três projetos, o "*OWASP Top 10 API*" (OWASP, 2022), "*OWASP Top 10 WEB* (OWASP, 2017b) e "*OWASP Mobile Top 10* (OWASP, 2021c), que são relatórios gerados a partir de pesquisas realizadas em entidades da comunidade de desenvolvimento de software, cujo objetivo é estabelecer um panorama sobre as dez vulnerabilidades mais recorrentes em aplicações. Vale destacar que esses relatórios ilustram as principais vulnerabilidades relevantes para o tema de pesquisa deste trabalho.

Figura 4 – Relatórios OWASP



Fonte: Adaptado de (OWASP, 2017a; OWASP, 2022)

Conforme reportado em OWASP (2022), muitos dos atuais serviços Web utilizam o protocolo REST (do inglês, *REpresentational State Transfer*) para interconexão entre serviços e clientes. Embora as APIs utilizem o mesmo protocolo HTTP (do inglês, *HyperText Transfer Protocol*) para comunicação com as aplicações Web, uma característica que diferencia as vulnerabilidades entre as APIs e as tradicionais aplicações *server-side* é o fato de as APIs não manterem estados com os clientes. A ausência de estado na comunicação entre diferentes tipos de aplicações pode levar a novas vulnerabilidades e tipos de ataques. Nessa direção, os relatórios OWASP (2017b) e OWASP (2022) corroboram com OWASP (2017a) no sentido em que as vulnerabilidades levantadas podem ser sanadas com revisão sistemática de código, uma técnica de engenharia de software que é aplicada após a implementação de um sistema. De acordo com

[1] https://owasp.org

Malek *et al.* (2019), a engenharia de software define seis atividades voltadas para a segurança de um sistema de software, a saber:

Especificação de requisitos de segurança. Os requisitos de segurança de um software podem ser divididos em: confidencialidade, integridade, disponibilidade e não repúdio. Além disso, os requisitos de segurança podem impactar diretamente sobre os requisitos não funcionais de um sistema. Mecanismos de criptografia e controles de acesso podem representar sobrecarga sobre o desempenho do sistema. Por exemplo, políticas de segurança que tornam as senhas de usuários mais fortes podem causar efeito adverso na usabilidade do sistema.

Modelagem de ameaças de aplicativos. Esta atividade visa identificar as eventuais ameaças que possam existir em um sistema de software específico, além de avaliar adequadamente tais ameaças, os riscos de segurança, e as implicações causadas em caso de sucesso de um determinado ataque. O sistema deve ter seus componentes ou arquitetura decompostos em alto nível sem preocupação com detalhes de baixo nível.

Desenvolvimento de arquitetura se segurança. A arquitetura de segurança é um conjunto de estratégias de mitigação e contramedidas incorporadas em um sistema, cujo propósito é reduzir os riscos de ameaças. Os componentes arquiteturais devem ser desenhados por meio de componentes, conectores, e associações entre os mesmos. As soluções de segurança resultantes da modelagem de ameaças são: (i) proteção, técnica empregada para proteger os ativos de ameaças; (ii) detecção, técnica empregada para monitorar e determinar a manifestação de uma ameaça; e, por fim, (iii) recuperação, técnica empregada para devolver o sistema a um estado de produção.

Validação e Verificação (V&V). Esta atividade tem por objetivo identificar as fraquezas de software, que podem ser sintetizadas em práticas de programação ruins. Tais práticas tendem a destacar estruturas ou comportamentos em software que são vulneráveis e exploráveis, os quais são conhecidos por contribuir negativamente com problemas de segurança. Portanto, pode-se dizer que, quando seguida, esta atividade auxilia na identificação e mitigação de ameaças de segurança mais comuns, melhorando significativamente a segurança de um sistema de software.

Teste de penetração. É uma atividade de natureza prática que visa revelar determinadas classes de vulnerabilidades, as quais podem residir dentro da superfície de ataque de uma aplicação, podendo ser alcançadas por meio de suas interfaces. Os testes de penetração podem ser classificados em: (i) caixa branca, onde as informações sobre o projeto e a implementação do sistema são usadas na geração de testes, ou (ii) caixa-preta, onde o engenheiro se limita essencialmente às mesmas informações disponíveis de um invasor.

***Patches* de software**: quando vulnerabilidades de segurança são encontradas em um sistema de software, os engenheiros desenvolvem e aplicam *patches* de segurança. Tais *patches* podem ser aplicados manualmente, implantando uma nova versão software ou dinamicamente em tempo de execução.

2.2.2 Conceitos e definições sobre autoproteção de sistemas

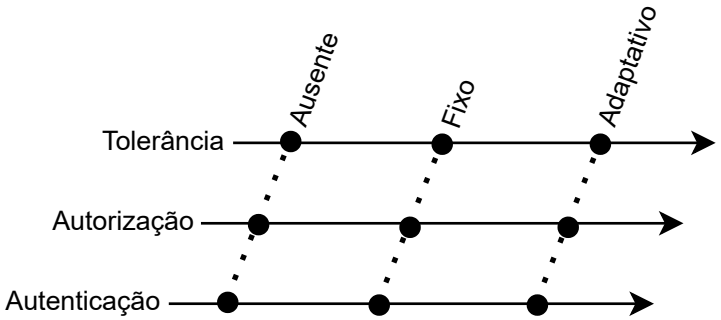
Na visão de Elkhodary e Whittle (2007), os sistemas de software contemporâneos operam em ambientes heterogêneos, dinâmicos e distribuídos, onde as necessidades de segurança mudam em tempo de execução. Devido ao aumento da demanda e complexidade dos ataques a sistemas de informação nos últimos anos, a manutenção manual por seres humanos a estes sistemas têm se tornado inviável por ser custosa e propensa a erros. Nesse sentido, sistemas que são capazes de lidar com segurança em tempo de execução podem se destacar em relação aos demais. Estes sistemas combinam conceitos e melhores práticas de engenharia para SaS (Seção 2.1), técnicas de inteligência computacional (Seção 2.3) e de segurança de sistemas (Seção 2.2.1).

Os sistemas de segurança adaptativa incluem um conjunto de serviços que devem ser implementados para satisfazer os objetivos de segurança do software, a saber: (i) autenticação é o processo de verificação da identidade fornecida pelo usuário, verificando se o mesmo é realmente quem afirma ser, além de impedir o envolvimento em atividades nas quais seus usuários participaram; (ii) autorização é o processo de controlar o acesso de usuários aos recursos do sistema que envolve a atribuição de privilégios e direitos de acesso aos recursos disponíveis; e, por fim, (iii) tolerância é o processo de tratar falhas em tempo de execução de um sistema, eliminando componentes defeituosos, limitando o efeito de falha e alocando substitutos válidos (ELKHODARY; WHITTLE, 2007).

A Figura 5 ilustra as dimensões de segurança para SaS, onde os serviços de tolerância, autenticação e autorização em um sistema podem variar em três níveis: (i) ausente de recursos de adaptação; (ii) fixo quanto ao nível de segurança e novas necessidades que podem surgir em tempo de execução; e (iii) adaptativo quanto ao processo de tomada de ações automáticas com o mínimo de participação humana para manter as especificidades de segurança. O serviço de tolerância atende ao princípio da disponibilidade, que visa garantir máxima continuidade e aceitabilidade do nível de serviço quando ocorrem falhas. Para isso, a autenticação permite que um sistema escolha entre diferentes métodos, com base em parâmetros específicos de contexto em tempo de execução, como ameaça do sistema ou nível de suspeita do usuário. Por fim, o serviço de autenticação permite que as políticas de controle de acesso sejam alteradas com base em parâmetros específicos do contexto.

De acordo com Tziakouris, Bahsoon e Babar (2018), os sistemas baseados em segurança adaptativa estão se tornando cada vez mais populares devido ao seu suporte potencial à

Figura 5 – Dimensões de segurança para SaS.



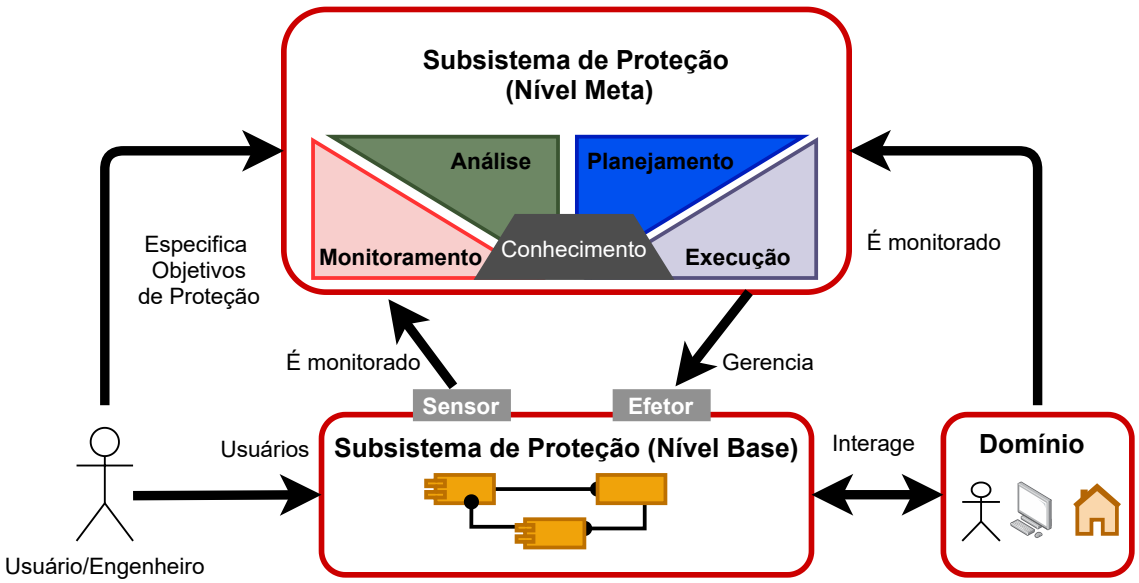
Fonte: Adaptado de (ELKHODARY; WHITTLE, 2007)

escalabilidade na detecção e mitigação de ameaças imprevistas em tempo de projeto que podem ser tratadas em tempo de execução. Esses sistemas têm como premissa principal diminuir a dependência de seres humanos para manutenção e tomadas de decisões. Portanto, a transparência operacional é um fator relevante, pois pode definir o nível de confiança dos usuários em relação a uma solução de segurança adaptativa. A transparência operacional varia em: (i) transparente, permite que usuários monitorem minuciosamente as atividades de adaptação; (ii) parcialmente transparente, apenas algumas das funcionalidades do sistema são visíveis para os usuários; e (iii) oculto, que não permite que usuários monitorem atividades de adaptação. Os autores supracitados comentam que as técnicas de detecção em sistema de autoproteção ou segurança adaptativa podem ocorrer por meio de técnicas de aprendizagem de máquina ou baseadas em modelos. As abordagens de aprendizado para detecção de comportamentos maliciosos podem variar em aprendizagem supervisionada e não supervisionada, permitindo que a detecção de ameaças ocorra de maneira proativa. Na detecção baseada em modelo são utilizadas assinaturas e regras que descrevem o comportamento normal ou mal-intencionado dos softwares, as quais são comparadas com o comportamento em tempo de execução para determinar se uma atividade é ou não mal-intencionado.

Yuan, Esfahani e Malek (2014) reportaram em seu estudo que a eficácia da autoproteção está relacionada ao ambiente de domínio no qual o software está inserido. Esse ambiente é composto por elementos que podem gerar impacto no software de nível básico, mas estão fora do domínio de controle exercido pelo subsistema de nível meta. Na Figura 6 é apresentada uma estrutura adaptativa para sistemas de segurança, onde informações do domínio são monitorados para ajudar na tomada de decisões de adaptação. Além disso, o sistema (Nível Meta) deve ser capaz de monitorar atividades externas no qual interagem com o sistema (Nível Base) para que seja possível tomar ações protetivas.

Por fim, Zseby, Pfeffer e Steglich (2009) apresentaram um relato sobre as quatro fases para autoproteção, a saber: prevenção, conscientização, tomada de decisão, e reconfiguração.

Figura 6 – Modelo de segurança adaptativa.



Fonte: (YUAN; ESFAHANI; MALEK, 2014)

A fase de **prevenção** é a primeira linha de defesa de um sistema e pode ser alcançada por meio de um "*design* de sistema" adequado quanto ao controle de acesso a recursos. O principal problema com a fase de prevenção é sua incapacidade de lidar com ataques desconhecidos ou imprevisíveis. A fase de **conscientização** está relacionada aos módulos de monitoramento e análise do modelo MAPE (IBM, 2005). Portanto, essa fase é responsável por detectar o ataque nas fases de preparação ou execução. Para isso, a seguinte sequência de atividades é executada: (i) *percepção*, observa fatos relevantes no ambiente; (ii) *inferência*, entende o significado dos fatos observados para a fase seguinte de detecção; e (iii) *previsão*, faz prognóstico dos próximos acontecimentos. A fase de **tomada de decisão** está relacionada ao planejamento do modelo MAPE-K (IBM, 2005), cujo objetivo é decidir se o sistema suspeita de um ataque ou "visualiza" uma falha que requer ação imediata. O mecanismo de detecção pode ser baseado em assinaturas (métodos estáticos) ou em anomalias (métodos dinâmicos). Por fim, na fase de **reconfiguração** ocorre a modificação do sistema, que é equivalente a fase de execução do modelo MAPE-K (IBM, 2005) para implantação das atividades de adaptação.

2.3 Inteligência computacional

Esta seção visa fornecer suporte para entendimento e/ou desenvolvimento de soluções para sistemas de autoproteção e/ou segurança adaptativa (veja Seção 2.2) que empregam técnicas de aprendizado de máquina para identificação de ataques/ameaças de segurança e sugestão para correção das mesmas. Para apresentar os assuntos supracitados, esta seção foi organizada da seguinte maneira: na Seção 2.3.1 são apresentadas definições sobre aprendizado de máquina,

além dos principais tipos de aprendizado, a saber: aprendizado supervisionado, aprendizado não supervisionado, aprendizado semi-supervisionado, e aprendizado por reforço; e por fim, na Seção 2.3.2 são apresentadas técnicas de inteligência computacional aplicadas ao contexto de segurança/cibersegurança.

2.3.1 Aprendizado de máquina

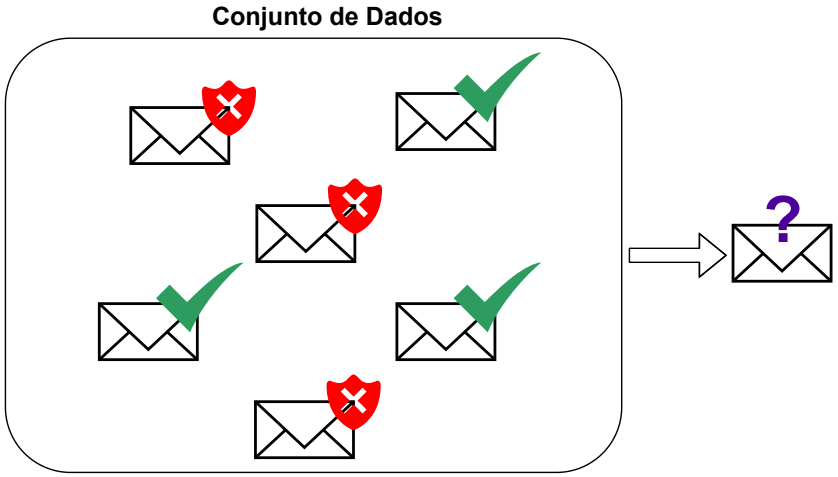
De acordo com Shalev-Shwartz (2014), aprendizado de máquina é a tarefa que converte experiência em especialização ou conhecimento. A experiência pode ser entendida como os dados de treinamento, os quais são insumos para os algoritmos de aprendizagem formar um conhecimento. Este, por sua vez, geralmente assume a forma de outro programa de computador que pode executar alguma tarefa. Em uma visão voltada a agentes, Norvig e Russell (2021) definiu que a aprendizagem ocorre à medida que um agente observa suas interações com o mundo e com seus próprios processos de tomada de decisão. Segundo Alpaydin (2014), a aprendizagem de máquina permite que engenheiros de software não tenham que prever e fornecer soluções para todas as situações possíveis. Dessa forma, baseados em amostras de dados, o sistema pode construir uma aproximação útil da realidade e, supondo que um futuro próximo não será muito diferente do passado, previsões futuras podem e devem estar corretas.

O campo de aprendizagem de máquina pode ser classificado em quatro abordagens de aprendizado, a saber: supervisionada, não supervisionada, semi-supervisionado, e por reforço (NORVIG; RUSSELL, 2021; GéRON, 2007). Mohri *et al.* (2012) reportou aprendizagem semi-supervisionada, inferência transdutiva, e aprendizagem *online* como outras abordagens de aprendizado, sendo destacado pelo autor que outros cenários de aprendizagem intermediários mais complexos podem ser encontrados. Géron (2007) cita ainda os critérios de aprendizagem em lote e *online*, onde é relevante se um sistema pode ou não aprender incrementalmente a partir de um fluxo de dados de entrada. No aprendizado em lote o treinamento ocorre utilizando todos os dados disponíveis em modo *offline*. Para que o sistema esteja apto a aprender com novos dados é necessário treinar uma nova versão do sistema com um conjunto de dados completos. Já no aprendizado *online*, o treinamento ocorre de maneira incremental, onde os dados de entrada são pequenos grupos chamados de "mini lotes". Este tipo de sistema geralmente é utilizado em situações que precisam se adaptar de maneira autônoma. A seguir, são descritas as abordagens de aprendizado supracitadas.

Aprendizado supervisionado. Esta abordagem envolve o aprendizado de uma função a partir de exemplos de suas entradas e saídas, sendo que os dados de entrada são rotulados com uma saída específica (NORVIG; RUSSELL, 2021). A Figura 7 ilustra um exemplo de aprendizado de máquina conhecido como classificação, onde mensagens de e-mail são classificadas como "válido" e "*spam*". O conjunto de dados geralmente é rotulado

por seres humanos, especialistas de domínio, e utilizado para alimentar o processo de treinamento, onde um algoritmo aprende a classificar tais mensagens. Após o processo de treinamento, pode-se dizer que existe um modelo capaz de classificar as novas mensagens de e-mail (GéRON, 2007).

Figura 7 – Aprendizado supervisionado por classificação



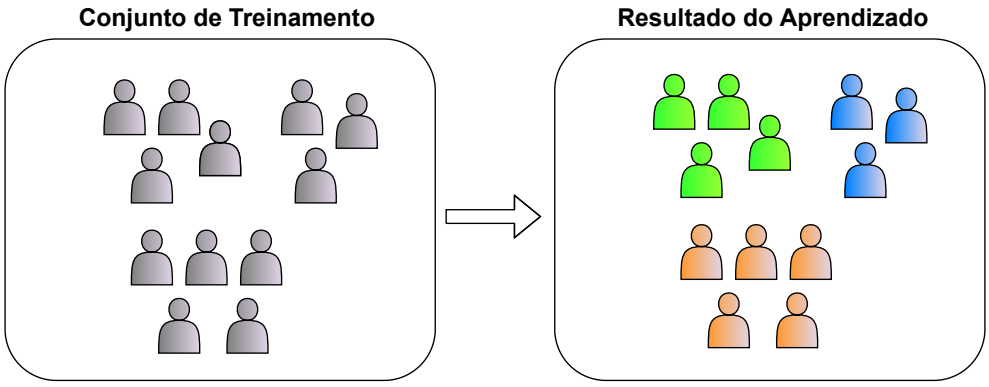
Fonte: Adaptado de (GéRON, 2007)

Enquanto os problemas de classificação tentam definir a probabilidade de um item pertencer a uma categoria como, por exemplo, um diagnóstico de câncer, outros problemas podem ser resolvidos por meio do método de regressão. O valor de venda de uma casa, o ângulo em que uma direção de carro autônomo deve ser rotacionada e a valorização de um imóvel são alguns exemplos de valores reais que esse tipo de método pode encontrar. Geralmente, nos problemas de regressão a resposta pretendida é um valor discreto (MOHRI *et al.*, 2012).

Por fim, vale destacar que no aprendizado supervisionado podem ser utilizados diversos algoritmos para treinamento, a saber: *k-Nearest Neighbors*, *Linear Regression*, *Logistic Regression*, *Support Vector Machines*, *Decision Trees*, *Random Forests*, *Neural networks*, entre outros (GéRON, 2007). Em resumo, esses algoritmos podem ser utilizados de modo individual ou de maneira combinada, sempre visando atender as necessidades/interesses de uma aplicação.

Aprendizado não supervisionado. Nesta abordagem, em oposição ao aprendizado supervisionado, os dados de treinamento não são rotulados, sendo o processo de aprendizado realizado sem a intervenção de um especialista (GéRON, 2007). Na Figura 8 é apresentado um exemplo de segmentação de *marketing*, onde usuários são agrupados de acordo com semelhanças em seus atributos. Nessa ilustração nota-se um conjunto inicial de usuários em cinza (caixa “Conjunto de Treinamento”) e o posterior agrupamento em cores verde, azul e laranja (caixa “Resultado de Aprendizado”).

Figura 8 – Aprendizado não supervisionado



Fonte: Adaptado de (GéRON, 2007)

Na visão dirigida por agentes de Norvig e Russell (2021), um agente de aprendizado baseado puramente nessa abordagem não pode aprender o que fazer porque não tem nenhuma informação sobre o que constitui uma ação correta ou um estado desejado. De acordo com Géron (2007), os algoritmos mais importantes para aprendizado não supervisionado são: *Clustering*, *Visualization and dimensionality reduction*, e *Association rule learning*.

Aprendizado semi-supervisionado. Nesta abordagem, o algorítimo de treinamento recebe um conjunto de dados em dois formatos de amostras: rotulados e não rotulados. A partir desse conjunto de dados são feitas previsões para todos os dados que não estejam rotulados com base naqueles que estão rotulados. Esta abordagem de aprendizado é utilizada em situações em que dados rotulados possuem alto custo, e dados não rotulados são de fácil acesso e baixo custo (MOHRI *et al.*, 2012). De acordo com Shah e Shah (2020), a quantidade de dados rotulados é muito menor em relação aos dados não rotulados. Portanto, um *cluster* de dados semelhantes é formado usando um algoritmo de aprendizado não supervisionado para que, em seguida, os dados rotulados sejam utilizados para rotular o restante dos dados não rotulados.

De acordo com Reddy, Viswanath e Reddy (2018), o principal objetivo do aprendizado semi-supervisionado é superar as desvantagens da aprendizagem supervisionada e não supervisionada. Os autores comentaram que o aprendizado supervisionado necessita de uma grande base de dados rotulada, o que exige a atuação de seres humanos (ou seja, especialistas de domínio), tornando assim os dados mais caros. Em contrapartida, o aprendizado não supervisionado possui baixa precisão no agrupamento dos dados. Ainda segundo os autores, aprendizagem supervisionada é subdividido em duas categorias: (i) classificação semi-supervisionada, onde pretende explorar informações presentes nos dados não rotulados para melhorar a qualidade classificador; e (ii) agrupamento semi-

supervisionado, onde informações parciais externas são usadas junto com informações dos dados rotulados para melhorar uma estrutura de grupos de dados.

Aprendizado por reforço. Esta abordagem é utilizada em situações em que uma única ação não possui relevância, mas sim uma sequência de ações corretas para atingir uma tarefa com sucesso. Nesse sentido, o programa de aprendizado de máquina deve ser capaz de avaliar a validade de uma sequência de ações e aprender com as sequências de boas ações anteriores para gerar uma política. Um exemplo de aplicação para este método de aprendizagem é um jogo de xadrez, onde um único movimento por si só não tem relevância, mas uma sequência de movimentações até chegar a um xeque mate é uma política relevante ao algoritmo de aprendizagem, sendo que o reforço ocorre ao final do jogo (ALPAYDIN, 2014). Em algumas situações, nenhum reforço pode ser fornecido pelo ambiente, então o agente se depara com o dilema, explorar ações desconhecidas para obter mais informações ou explorar as informações já coletadas (MOHRI *et al.*, 2012).

Um agente de aprendizagem por reforço, em vez de ser informado sobre o que fazer por um instrutor, deve aprender a partir de recompensas, as quais retroalimentam o agente para novas interações com o ambiente em busca de aprendizagem (NORVIG; RUSSELL, 2021). Para Géron (2007), o aprendizado por reforço ocorre através de um agente, que por meio de políticas, deve aprender por si mesmo qual deve ser a estratégia a ser adotada para obter a maior recompensa possível ao longo do tempo. Para isso, o agente deve observar o ambiente em que está inserido, selecionar e executar ações a partir de políticas e de suas experiências.

2.3.2 Ciência dos dados e cibersegurança

De acordo com Aftergood (2017), o termo cibersegurança se refere ao conjunto de tecnologias e processos aplicados na proteção de recursos de tecnologias da informação contra ataques ou acessos não autorizados. Esta seção apresenta conceitos e as principais metodologias para se trabalhar com ciência dos dados na extração de conhecimento de segurança para ser utilizado na classificação, predição e detecção de ataques e ameaças. Para isso, inicialmente são apresentadas as principais metodologias para aplicação de ciência dos dados, além de uma metodologia genérica definida por Sarker *et al.* (2020). Em seguida, alguns conjuntos de dados relevantes para extração de conhecimento no contexto da segurança são apresentados.

Metodologias para ciência dos dados. A ciência de dados engloba a extração de informações e a descoberta de conhecimento de dados utilizando abordagens científicas. Estas metodologias podem desempenhar um papel significativo na área de cibersegurança, utilizando o poder dos dados, computação de alto desempenho, mineração de dados, e aprendizado de máquina (FOROUGHI; LUKSCH, 2018). Na visão deste autor, o processo de extração de

conhecimento pode ser sintetizado em seis passos: seleção, pré-processamento, transformação, mineração de dados, interpretação e avaliação. Neste mesmo estudo, os autores apresentam e comparam quatro metodologias populares de ciência de dados para abordagens de cibersegurança. A seguir são abordados cada uma das metodologias (FOROUGHI; LUKSCH, 2018).

***Knowledge-Discovery in Databases* (KDD)**. É uma metodologia que utiliza técnicas de mineração de dados para extrair conhecimento de banco de dados utilizando medidas e limites específicos. Em síntese, esta metodologia é composta pelos seguintes passos: (i) seleção, gera ou produz um conjunto de dados de destino, que se concentra em um subconjunto de variáveis ou amostra de dados em um banco de dados; (ii) pré-processamento, obtêm dados consistentes limpando ou preprocessando os dados selecionados; (iii) transformação, reduz a dimensionalidade do recurso usando métodos de transformação de dados; (iv) mineração de dados, tenta reconhecer padrões de atenção ou comportamentos usando técnicas de mineração de dados; e, (v) avaliação e interpretação, o padrão minerado na fase anterior deve ser avaliado e interpretado.

***Cross Industry Standard Process for Data Mining* (CRISP-DM)**. Esta metodologia é organizada em seis fases de alto nível que descrevem o processo analítico. Embora tais fases sejam bem definidas e documentadas, é necessário que seja atualizado para atender as especificidades de projetos de cibersegurança. Essas fases podem ser sintetizadas em: (i) compreensão do negócio, nesta fase ocorre a compreensão do problema ou requisito de projeto, que é transformada em problema de mineração de dados; (ii) entendimento de dados, é criado um conjunto de dados brutos no qual identifica subconjuntos de dados para desenvolver hipóstases, dados esses que iniciam em uma fase de coleta de dados; (iii) preparação de dados, é construído um conjunto de dados final com base nos dados brutos da fase anterior; (iv) modelagem, técnicas e estratégias de modelagem são selecionadas e aplicadas; (v) avaliação, modelos são selecionados a partir dos quais foram obtidos na fase anterior, onde estes garantem a generalização em relação aos dados ocultos; e (vi) implantação, fase de implantação de uma representação de código do modelo ou modelos finais obtido na fase de avaliação.

***Foundational Methodology for Data Science* (FMDS)**. Metodologia composta por fases semelhantes às duas anteriores, porém com práticas adicionais como uso de volumes de dados grandes, análises de textos e imagens, *deep-learning*, e processamento de linguagens. A metodologia é composta por 10 etapas iterativas, a saber: (i) compreensão do negócio, busca uma resolução lucrativa e eficaz de um problema de negócio; (ii) abordagem analítica, identifica a técnica de aprendizado de máquina adequada para resolver o problema e obter o resultado desejado; (iii) requisitos de dados, define os requisitos de dados necessários com base na abordagem analítica escolhida na fase anterior; (iv) coleta de dados, são

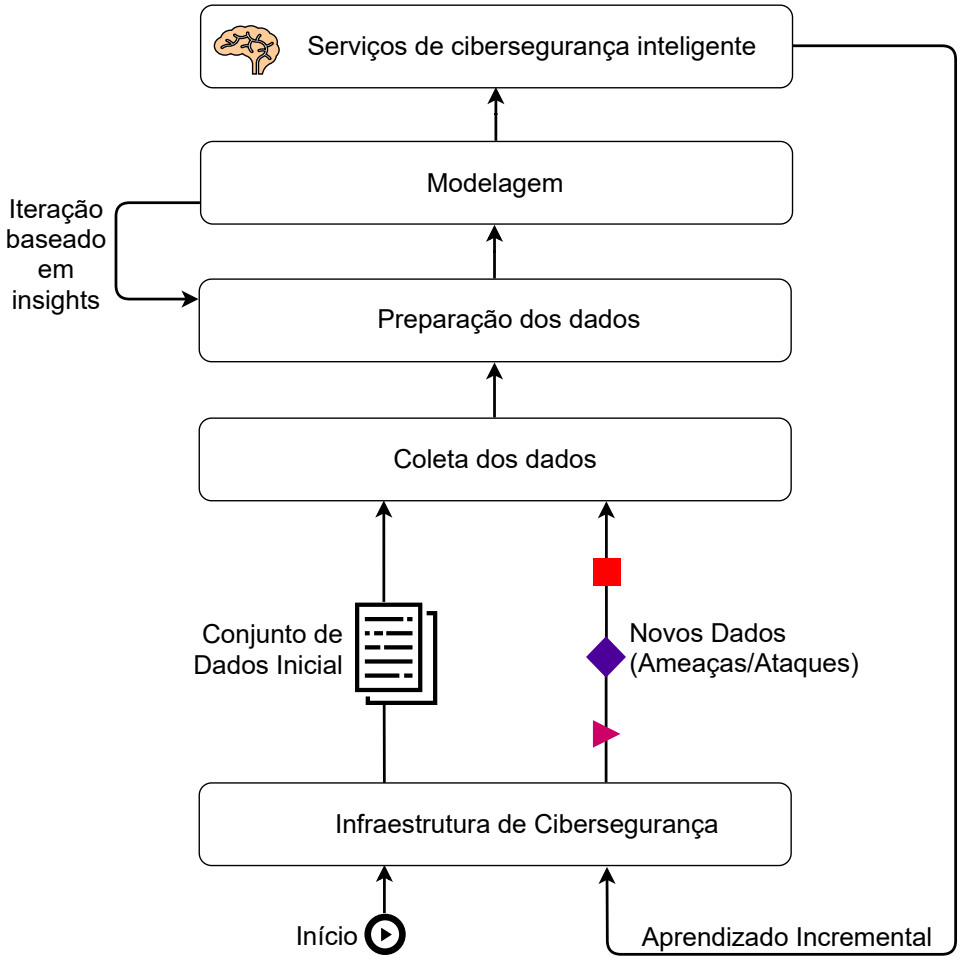
identificados e coletados os recursos de dados disponíveis relacionados e aplicáveis ao domínio do problema; (v) compreensão de dados, através de estatísticas descritivas e métodos de visualização são compreendidos o conteúdo dos dados, avaliada a qualidade dos dados e explorada as percepções dos dados; (vi) preparação de dados, compreende todas as tarefas de construção do conjunto de dados que serão utilizados na fase de modelagem; (vii) modelagem, esta fase se concentra no desenvolvimento do modelo preditivo ou descritivo com base na abordagem analítica descrita na fase (vi) e usando a primeira versão do conjunto de dados preparado como um conjunto de treinamento; (viii) avaliação, avalia a qualidade e eficácia do modelo desenvolvido para perceber se o mesmo aborda de maneira completa e apropriada o problema da segurança cibernética; (ix) implantação, depois que o modelo desenvolvido for aprovado e credenciado na fase de avaliação (ou seja, cobre o desafio da segurança cibernética de maneira adequada), o mesmo deve ser implantado no ambiente de produção ou de teste comparável; e (x) *feedback*, coleta os resultados da edição implementada do modelo analítico para analisar e fazer o *feedback* de sua funcionalidade, desempenho e eficiência de acordo com o ambiente de implantação.

***Team Data Science Process* (TDSP)**. É uma metodologia que tem o objetivo de fornecer análises preditivas, que incluem inteligência artificial e aprendizado de máquina. Em síntese, essa metodologia é composta pelos seguintes passos: (i) compreensão do negócio, onde é definido o objetivo do problema, o modelo preditivo a ser utilizado, e a fonte de dados necessária; (ii) aquisição e compreensão de dados, onde os dados são coletados, transferidos para o local de trabalho, e limpos. Ao final desta etapa, pode ser necessário voltar à primeira fase para coletar mais dados; (iii) modelagem, onde são realizados os processos de engenharia de recursos, treinamento do modelo, seleção de algorítimo, criação de modelo e avaliação de modelo preditivo. Ao final, os dados coletados são divididos em conjuntos de dados de treinamento e testes para que o modelo de aprendizado possa ser treinado e avaliado; (iv) implantação, onde é gerado um modelo de análise em tempo real ou em lote, onde o produto de dados final deve ser credenciado pelo cliente; e (v) aceitação do cliente, onde o cliente deve confirmar o *pipeline* de dados, modelo preditivo e implantação do produto.

Por fim, no trabalho de Sarker *et al.* (2020), uma metodologia genérica de ciência de dados voltado para a cibersegurança é proposta com base em quatro fases: (i) coleta de dados, nesta fase são coletadas informações a partir do domínio que se deseja utilizar no contexto de segurança. As fontes dos dados podem ser de rede, *host* ou híbrida; (ii) preparação dos dados, os dados brutos obtidos na primeira fase são preparados visando construir o modelo de aprendizagem. Nesta fase, dados inúteis são removidos, ruídos são limpos, dados podem ser convertidos, obtendo ao final desta etapa somente os dados normatizados; (iii) modelagem, nesta fase ocorre a extração

de conhecimento dos dados processados, processo que compreende principalmente no uso de aprendizagem de máquina e engenharia de recursos; e (iv) aprendizagem incremental e dinâmica, o modelo gerado na fase anterior teve como base a utilização de dados estáticos da fase inicial, e para que o modelo se mantenha atualizado é necessário incorporar novos conhecimentos. Para evitar o processamento de todos os dados novamente, os novos dados devem ser inseridos e processados de maneira incremental. A Figura 9 ilustra as fases modelo em camadas apresentado.

Figura 9 – *Framework* de IA para Cibersegurança



Fonte: Adaptado de (SARKER *et al.*, 2020)

Conjunto de dados para cibersegurança. O elemento central na ciência de dados é o conjunto de dados utilizado para treinamento de algorítimos. Existem diversos conjuntos de dados disponíveis para o treinamento de algoritmos no contexto de cibersegurança. Porém, de acordo com Sarker *et al.* (2020), alguns dos conjuntos disponíveis são antigos e podem não refletir bons treinamentos. Além disso, modelos de classificação não apresentam uma generalização satisfatória para diferentes conjuntos de dados (SARKER *et al.*, 2020). No trabalho apresentado por Bhagwani *et al.* (2019), os autores utilizaram o conjunto de dados "CSIC 2010 HTTP" para detectar anomalias em tráfego Web e conseguiram uma precisão de acerto de 99,94%. Ao tentar

aplicar o mesmo modelo em um conjunto de dados próprio, a melhor precisão foi de 88,84%. A seguir, são apresentadas alguns conjuntos de dados e suas principais características:

HTTP CSIC Torpeda 2012 Dataset. Esta base foi criada a partir do *framework* Torpeda, cujo objetivo é gerar tráfego Web rotulado (BHAGWANI *et al.*, 2019). Essa base possui 74.133 registros de solicitações para *e-commerce*, sendo que 8.363 válidas, 16.459 anômalas, e 49.311 maliciosas (TORRANO-GIMENEZ; PEREZ-VILLEGAS; ALVAREZ, 2012).

ECML/PKDD 2007 Dataset. Esta base possui classificações de solicitações HTTP mal-intencionadas recebidas por um aplicativo Web e a identificação das solicitações de ataque. Em números, essa base possui 35.006 registros de solicitações válidas e 15.110 registros de tentativas de ataques (BHAGWANI *et al.*, 2019).

KDD Cup 1999 Dataset. Esta base foi formada pelo dados obtidos no *MIT Lincoln Labs*, que são utilizados geralmente para avaliação de sistemas de detecção de intrusão. Além disso, essa base também contém um conjunto padrão de dados a serem auditados, que inclui uma ampla variedade de intrusões simuladas em um ambiente de rede militar (SALLOUM *et al.*, 2020).

ISOT Dataset. Esta base representa um conjunto de dados que possui um total de 1.675.424 registros fluxo de tráfego que aborda os *botnets Storm* e *Waledac* (SALLOUM *et al.*, 2020).

HTTP CSIC 2010 Dataset. Esta base é formada por dados rotulados em "anômalo" e "normal", os quais foram gerados pelo Conselho Nacional de Pesquisa da Espanha (SALLOUM *et al.*, 2020). Os registros retratam o tráfego de aplicações Web, sendo quase 6.000 solicitações rotuladas como normais e mais de 25.000 como anômalas (SALLOUM *et al.*, 2020).

2.4 Considerações finais

Neste capítulo foram apresentados na Seção 2.1 os conceitos de SaS, área de pesquisa de grande importância para os sistemas de software atuais, especialmente para o contexto deste trabalho. O conteúdo da referida seção é fundamental para entendimento da Seção 2.2, que aborda a autoproteção de sistema e segurança adaptativa, uma mescla de duas áreas de pesquisa, SaS e segurança de sistemas. Na Seção 2.2.1 foram apresentados os principais conceitos a respeito segurança de aplicações Web. Além disso, foram apesentadas as dez vulnerabilidades de maior ocorrência em aplicações Web, API e móvel, conforme pesquisas realizadas na academia e indústria de software. Por fim, na Seção 2.3 foram apresentados conceitos e definições sobre aprendizado de máquina e as principais metodologias aplicadas aos domínios de aplicações baseadas em segurança/cibersegurança, os quais são de grande relevância no contexto de autoproteção de sistemas.

3 AUTOPROTEÇÃO EM APLICAÇÕES BASEADAS EM SERVIÇOS

O objetivo deste capítulo é apresentar um panorama sobre o projeto de abordagens de autoproteção (do inglês, *self-protecting*) para SApps. Sistemas que implementam mecanismos de autoproteção pode ser considerado como uma classe especial de sistemas de software, pois permitem reconhecer ameaças/ataques que estejam ocorrendo em tempo de execução e propor soluções para que tais sistemas não sejam comprometidos. Na literatura, sistemas baseados em autoproteção são objetos de pesquisa nas comunidades científicas de computação autônoma e SaS. Além disso, nota-se também que esse termo pode ser considerado um sinônimo para sistemas que lidam com segurança em tempo de execução, uma outra comunidade científica conhecida como segurança adaptativa. Como este trabalho está inserido dentro de um contexto de pesquisa voltado para a comunidade de SaS, apenas o termo autoproteção será utilizado como referência deste ponto em diante. Como forma de apresentar os assuntos relacionados ao tema deste trabalho, este capítulo foi organizado da seguinte maneira: na Seção 3.1 é discutida uma investigação preliminar sobre os estudos secundários (ES) que envolvem abordagens de autoproteção para SApps; na Seção 3.2 é apresentado o mapeamento da literatura sobre o tema de pesquisa deste trabalho; e, por fim, na Seção 3.4 são reportadas as considerações finais deste capítulo.

3.1 Estudos secundários sobre abordagens de autoproteção para SApps

Técnicas de revisão da literatura permitem sistematizar um conhecimento, ou seja, resumir, avaliar e interpretar a relevância de todas as evidências relacionadas a uma questão específica, área temática ou fenômeno de interesse. Um estudo individual é chamado de "estudo primário", ao passo que o resultado de uma revisão sistemática é chamado de "estudo secundário" (KITCHENHAM; CHARTERS, 2007; KITCHENHAM *et al.*, 2010; PETERSEN *et al.*, 2008; PETERSEN; VAKKALANKA; KUZNIARZ, 2015). Esta seção apresenta uma investigação por estudos secundários no contexto de abordagens de autoproteção para SApps. Visando nortear as atividades de pesquisa deste capítulo, um protocolo (ver Apêndice A) foi elaborado cobrindo cinco passos para esse tipo de trabalho (PETERSEN *et al.*, 2008; PETERSEN; VAKKALANKA; KUZNIARZ, 2015): (i) definição das questões de pesquisa; (ii) condução da pesquisa; (iii) triagem dos artigos; (iv) elaboração dos resumos; e (v) extração de dados e

mapeamento dos estudos.

Para conduzir uma investigação voltada para recuperar estudos secundários sobre abordagens de autoproteção para SApps, os termos definidos no protocolo de pesquisa (ver Tabela 7) foram combinados aos seguintes termos: SLR, SMS, *survey*, *systematic literature review*, *systematic mapping*, *systematic mapping study*, *systematic review*. A Tabela 1 mostra a *string* de busca resultante dessa combinação.

Tabela 1 – *String* de pesquisa para estudos secundários

({SLR} OR {SMS} OR {survey} OR {systematic literature review} OR {systematic mapping} OR {systematic mapping study} OR {systematic review})
AND
({adaptive security} OR {autonomous security} OR {self-protecting} OR {self-protection} OR {self-safety} OR {self-securing} OR {self-security})
AND
({analytics} OR {big data} OR {computational intelligence} OR {data mining} OR {data science} OR {KDD} OR {learning} OR {natural language processing} OR {reinforcement} OR {semi-supervised} OR {supervised} OR {unsupervised})

Para selecionar os estudos secundários relevantes, os critérios apresentados na Seção A.3 foram adaptados para este tipo de investigação. Assim, o critério de inclusão (CI) e critérios de exclusão (CE) para seleção estudos secundários são:

- **CI1**: O estudo é um trabalho secundário que aborda abordagem de autoproteção para SApps;
- **CE1**: O estudo não aborda abordagem de autoproteção para SApps;
- **CE2**: O estudo secundário é um editorial, documento de posição, *keynote*, opinião, tutorial, pôster ou painel;
- **CE3**: o estudo secundário foi escrito em um idioma diferente do inglês;
- **CE4**: O estudo secundário tem apenas um resumo ou não está disponível na íntegra; e
- **CE5**: O estudo secundário é um trabalho anterior desenvolvido pelo mesmo autor. Nesse caso, apenas o estudo mais recente ou mais completo é considerado.

A busca teve como referência as mesmas bases de pesquisa apresentadas na Tabela 6. Essa atividade foi conduzida no período de 04/01 a 25/01/2021, resultando 540 estudos secundários.

Os estudos resultantes passaram por um processo de filtragem organizado em três fases, a saber: na **Fase 0** são apresentados os números obtidos em cada base; na **Fase 1** os estudos foram agrupados em um arquivo para facilitar a identificação de duplicatas; na (**Fase 2**) foram removidos 35 ocorrências de estudos em duplicidade; e, por fim, na (**Fase 3**) foram aplicados os critérios de seleção (**CI** e **CE**). Os números dessas fases podem ser observados na Tabela 2.

Tabela 2 – Lista de bases de pesquisa

Base de pesquisa	**Fase 0**	**Fase 1**	**Fase 2**	**Fase 3**
ACM Digital Library	133	540	505	4
IEEE Xplore	2			
ScienceDirect	58			
Scopus	12			
Springer Link	327			
Web of Science	8			

Como pode-se observar na Tabela 2, quatro estudos secundários que abordam autoproteção para SApps foram selecionados. A Tabela 3 mostra as referências dos estudos ordenadas pelo título, sendo apresentado o identificador (coluna ID) e a referência completa de cada estudo (coluna Referência). A seguir, uma síntese de cada estudo é reportada.

Tabela 3 – Trabalhos selecionados

ID	**Referência**
ES1	Arias-Cabarcos, P., Krupitzer, C. and Becker, C., **A Survey on Adaptive Authentication**, ACM Comput. Surv., Association for Computing Machinery, 2019, Vol. 52(4).
ES2	Tziakouris, G., Bahsoon, R. and Babar, M. A., **A Survey on Self-Adaptive Security for Large-Scale Open Environments**, ACM Comput. Surv., Association for Computing Machinery, 2018, Vol. 51(5).
ES3	Yuan, E., Esfahani, N. and Malek, S., **A Systematic Survey of Self-Protecting Software Systems**, ACM Trans. Auton. Adapt. Syst., Association for Computing Machinery, 2014, Vol. 8(4).
ES4	Yuan, E. and Malek, S., **A Taxonomy and Survey of Self-Protecting Software Systems**, Proceedings of the 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, IEEE Press, 2012, pp. 109–118.

Arias-Cabarcos, Krupitzer e Becker (2019) apresentaram uma revisão sistemática sobre autenticação adaptativa que apontou os principais desafios e direções futuras relacionados ao tema. De acordo com esses autores, os atuais sistemas de autenticação adaptativa são difíceis de estender e reutilizar, o que tem dificultado o desenvolvimento de novas soluções. Nesse sentido,

por meio de uma taxonomia, o trabalho identificou problemas de *design* e apresentou um roteiro que permite o desenvolvimento de novas soluções dinâmicas e adaptativas. Os autores também comentaram que a maioria dos trabalhos avaliados utilizam senhas e PINs (do inglês, *Personal Identification Number*) como meios de autenticação. Em um cenário paralelo, o estudo revelou que alguns trabalhos abordam mecanismos baseados em biometria e comportamentos como alternativas de autenticação mais transparentes aos usuários. No que diz respeito ao conteúdo, essa taxonomia foi organizada em quatro itens: (i) "o que adaptar", sendo o que deve ser modificado no software, que pode ser por meio de parâmetro ou alterações estruturais; (ii) "por que adaptar", sendo o motivo que origina a adaptação; (iii) "como adaptar", onde são abordados os controles do processo de adaptação; e (iv) "quando e onde adaptar", ou seja, as nuances operacionais do processo de adaptação. Por fim, como direções para pesquisas futuras, segundo os autores é provável que os atuais meios de autenticação não sejam substituídos por um novo mecanismo, mas sim por um mecanismo dinâmico que selecione o melhor meio de autenticação baseado na situação/ambiente do sistema e do usuário. Além disso, também é destacado nesse estudo que os sistemas de autenticação adaptativa atuais não foram projetados com base em abordagens metodológicas e/ou boas práticas de engenharia de software, o que pode explicar a dificuldade de evolução desse tema de pesquisa.

Em Tziakouris, Bahsoon e Babar (2018) os autores apresentaram uma taxonomia centrada em arquiteturas para ambientes abertos e ultra-grandes voltados para aplicações que utilizam segurança autoadaptativa. De acordo com esses autores, existem poucos estudos acadêmicos nesta temática de pesquisa, sendo os mesmos mapeados e comparados a fim de identificar as direções futuras de pesquisa nesta temática. A taxonomia proposta pelos autores supracitados consiste em 16 dimensões na qual se enquadram em dois grupos, modelo conceitual e características de adaptação. Para trabalhos futuros foram identificados algumas tendências, a saber: (i) aplicar mecanismos proativos de segurança; (ii) projetar mecanismos que permitam alternar entre diferentes arquiteturas com base nas mudanças nos requisitos de segurança; (iii) avançar na segurança centrada em ativos, permitindo que os usuários apliquem diferentes políticas de segurança em diferentes ativos; (iv) explorar a ligação entre os componentes de alto e baixo nível dos sistemas de segurança adaptativa; (v) explorar segurança voltada a computação orientada a serviços; e (vi) criar repositórios públicos para manter serviços e recursos de segurança que possam ser acessados sob demanda pelos sistemas.

Yuan, Esfahani e Malek (2014) desenvolveram uma taxonomia a partir de uma revisão sistemática e de experiências anteriores (YUAN; MALEK, 2012), cujo objetivo foi classificar e caracterizar sistemas que abordam a autoproteção. A revisão sistemática partiu de investigações de ameaças que são cada vez mais recorrentes ao período de investigação, além de comportamentos arquiteturais dinâmicos para sistemas dessa natureza. A taxonomia resultante do trabalho mostrou que o tema autoproteção deve avançar nas seguintes áreas: (i) mudar o foco das camadas de rede

e *host* para abordagens independentes de camadas e baseadas em arquitetura, e (ii) segurança de perímetro até a proteção geral do sistema. Além disso, o estudo identificou direções futuras de pesquisa no contexto de autoproteção como, por exemplo, direcionar esforços para proteger o módulo de gerenciamento de autoproteção, manter uma separação clara dos módulos de gerenciamento e domínio (ou seja, sistemas que monitorem todos os módulos de um sistema), e desenvolvimento de técnicas de segurança reativa e proativa.

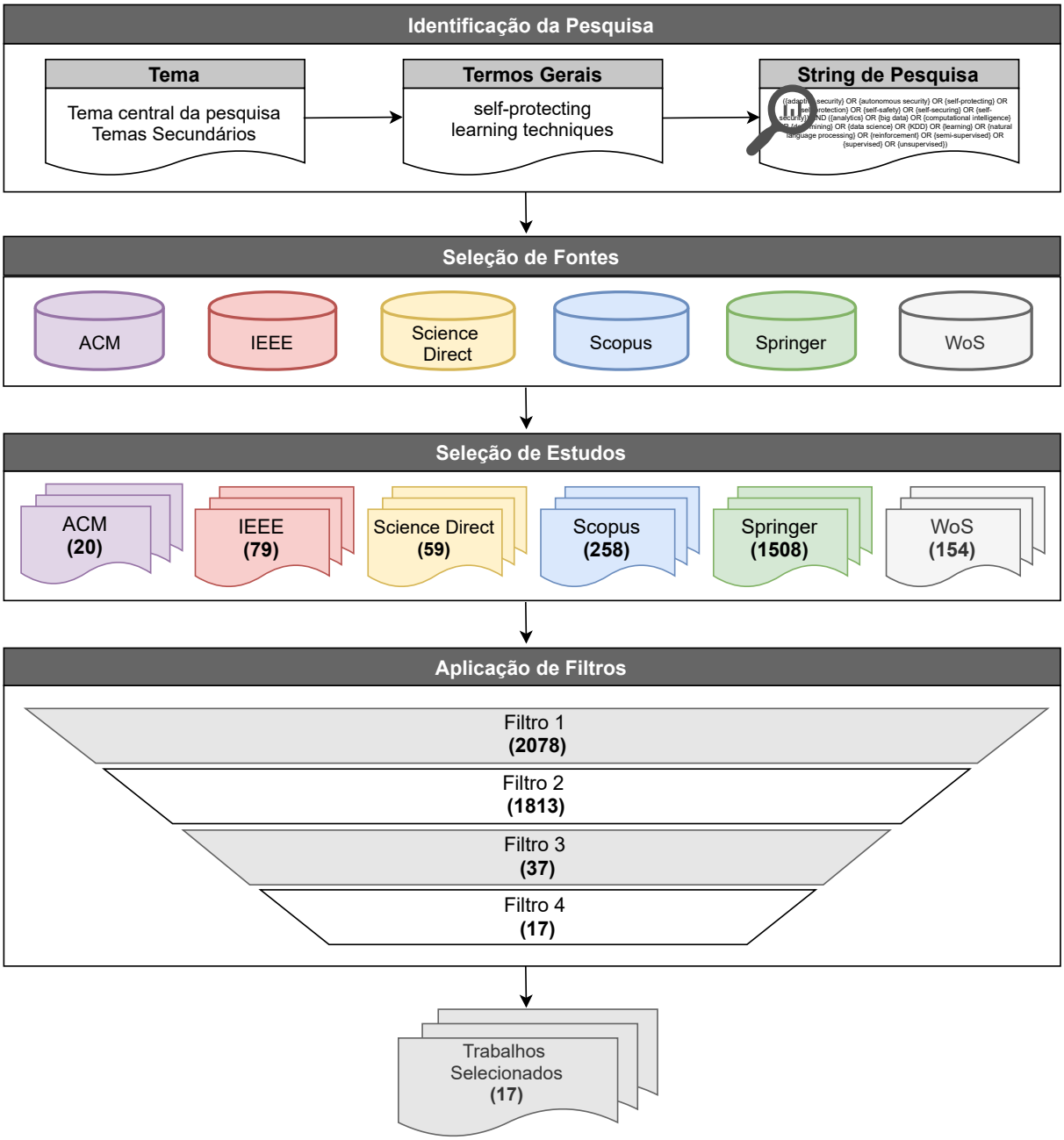
Uma taxonomia para sistemas de software no contexto de autoproteção foi desenvolvido por Yuan e Malek (2012). O objetivo dessa taxonomia é apoiar a classificação e caracterização de ameaças. Além disso, os autores também descreveram suas experiências com a aplicação da taxonomia em várias abordagens existentes. A taxonomia investiga ameaças e comportamentos arquiteturais dinâmicos no qual contribui com um extenso levantamento do estado da arte da autoproteção de sistemas de software, além de identificar padrões e lacunas existentes nessa área de pesquisa. O trabalho reporta recomendações para direções futuras de pesquisa voltadas para sistemas de autoproteção, tais como: (i) buscar abordagens mais "integradas" que abrangem a fase de *design* e o tempo de execução; (ii) explorar abordagens mais descentralizadas de coordenação, planejamento e otimização; (iii) explorar medidas qualitativas e quantitativas que podem ser usadas para avaliar dinamicamente a postura geral de segurança do sistema; (iv) buscar abordagens de proteção e monitoramento geral do sistema; e (v) catalogar e automatizar padrões de adaptação de segurança no nível de arquitetura abstrata.

Diante do exposto, os quatro estudos secundários existentes na literatura podem apoiar o projeto de abordagem de autoproteção para SApps. Como pode-se observar, tais estudos reportam diferentes aspectos de investigação que têm se mostrado de interesse pela academia e indústria de software. Embora esses estudos tenham sido importantes para proposição de novas soluções de autoproteção, uma investigação mais ampla e direcionada a outros assuntos que permeiam o projeto de abordagens de autoproteção para SApps foi conduzida neste trabalho. Dessa forma, na Seção 3.2 são reportados os detalhes do mapeamento da literatura, assim como os resultados e contribuições do mesmo para o desenvolvimento da proposta deste projeto de mestrado acadêmico.

3.2 Mapeamento sistemático sobre abordagens de autoproteção para SApps

Esta seção tem por objetivo apresentar uma revisão da literatura sobre abordagens de autoproteção para SApps conduzido neste trabalho. Para isso, um mapeamento sistemático (do inglês, *Systematic Mapping Study* – SMS) foi conduzido, visando estabelecer um panorama geral sobre o tema de pesquisa deste projeto (PETERSEN *et al.*, 2008; PETERSEN; VAKKALANKA;

Figura 10 – Mapeamento Sistemático



Fonte: Autoria própria

KUZNIARZ, 2015). De acordo com Petersen *et al.* (2008), Petersen, Vakkalanka e Kuzniarz (2015), um SMS permite apresentar uma visão justa e completa sobre um determinado tema da literatura mitigando/minimizando possíveis influências que um pesquisador possa sofrer. O SMS apresentado nesta dissertação permitiu verificar, em diferentes bases de publicação, a existência de estudos primários relacionados ao tema de pesquisa deste trabalho (ou seja, abordagens de autoproteção para SApps). Para isso, um protocolo de pesquisa foi elaborado (veja Apêndice A), onde são apresentados os detalhes de como esse mapeamento foi conduzido. A Figura 10 apresenta o processo de SMS organizado em 4 fases, as quais são detalhadas a seguir:

Identificação da Pesquisa. Nesta fase é definido o protocolo de pesquisa do mapeamento (veja Apêndice A), onde são definidos os seguintes itens: tema de pesquisa, termos comuns sobre o tema, bases de pesquisa que utilizadas no mapeamento, e *string* de busca. Como o tema deste trabalho envolve a combinação de diferentes áreas de pesquisa, optou-se pela seguinte organização: (i) tema central de pesquisa, composto de "segurança adaptativa/abordagens de autoproteção" e (ii) temas secundários, "computação orientada a serviços", "sistemas autoadaptativos", "técnicas de inteligência computacional", e "computação móvel". Tendo como referência que a autoproteção pode ser aplicada em sistemas Web, móveis e baseados em APIs (veja Seção 2.2), o interesse deste mapeamento é entender como as soluções de autoproteção têm sido projetadas nesses últimos anos para esses tipos de aplicação. Para isso, os termos que representam autoproteção ("self-protecting") e técnicas de aprendizado ("learning techniques") foram considerados como os principais desse mapeamento. Esses termos foram definidos em função das pesquisas preliminares reportadas na Seção 3.1, onde foi possível identificar o estado da arte dessa área de pesquisa, assim como suas lacunas.

Seleção de fontes. Nesta fase são selecionadas bases de pesquisa, as quais são recomendadas por Dyba, Dingsoyr e Hanssen (2007), Kitchenham e Charters (2007), Petersen, Vakkalanka e Kuzniarz (2015) para condução de trabalhos dessa natureza na área de engenharia de software.

Seleção de Estudos. Nesta fase foi utilizada a *string de busca* para recuperação dos estudos nas bases de pesquisas. Sobre esta fase, vale destacar que os anos de publicação foram limitados aos últimos 4 anos (ou seja, de 2018 a 2021), sendo os três primeiros em anos inteiros e o último em fração pela data de execução das buscas. Por fim, cabe ressaltar que tal decisão foi tomada em função da área de cibersegurança ser extremamente perecível, onde técnicas de defesas amplamente adotadas se tornam obsoletas com as novas ameaças/vulnerabilidades que surgem diariamente. Esse fator tem sido mencionado por diversos autores como sendo a maior dificuldade para consolidação dessa área de pesquisa (veja Seção 3.1).

Aplicação de Filtros. A partir dos estudos selecionados na fase de "Seleção de Estudos", quatro filtros foram aplicados com base nos critérios de seleção definidos no protocolo de pesquisa (veja Seção A.3). Esse processo pode ser sintetizado da seguinte maneira: (i) "Filtro 1", os trabalhos obtidos das bases de pesquisa foram unificados em um único arquivo em formato `.bib`, que resultou em 2.078 estudos; (ii) "Filtro 2", os trabalhos duplicados foram removidos, resultando em 1.813 estudos; (iii)"Filtro 3", foram aplicados os CI e CE nos títulos, palavras chaves e resumo dos estudos, que resultou em 37 estudos; e, por fim, (iv) "Filtro 4", os CI e CE foram aplicados após a leitura na íntegra de cada estudo. Dessa forma, para que um estudo fosse incluído no conjunto de estudos final do mapeamento, o

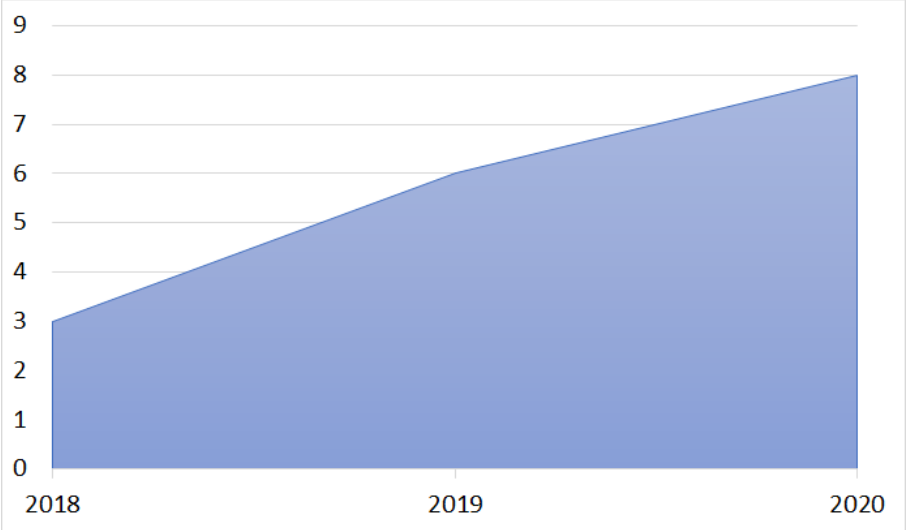
mesmo deve abordar simultaneamente técnicas de inteligência computacional aliada ao uso de autoproteção. Como resultado, 17 estudos foram selecionados para o mapeamento que será apresentado nesta seção. A lista completa desses estudos pode ser visualizada na Tabela 9, a qual foi utilizada para extração de informações e geração de todas estatísticas e conteúdo que serão utilizados para responder às questões de pesquisa estabelecidas no protocolo de pesquisa (veja Seção A.2).

A seguir serão apresentados os dados extraídos de cada QP (Questão de Pesquisa) do mapeamento, sendo que os dados serão apresentados em porcentagem e formatados com duas casas decimais. Portanto cabe ressaltar que em algumas QPs, a somatória das porcentagens não totalizam 100% devido ao arredondamento dos números. Antes de iniciar a apresentação do conteúdo que responde a cada QP do mapeamento, três cenários de informação, que são comuns em trabalhos de revisão da literatura, devem ser reportados. O primeiro ilustra a distribuição dos estudos durante o período coberto por este mapeamento, como pode-se observar na Figura 11. Com base no gráfico exposto, observa-se que houve um crescente aumento na quantidade de trabalhos publicados nos últimos anos, considerando o período de 2018 a 2020. Em relação ao gráfico apresentado, vale destacar que essa distribuição é relacionada especificamente a estudos que apresentaram soluções para autoproteção de sistemas que abordaram simultaneamente alguma técnica de inteligência computacional aplicada ao contexto de autoproteção sistemas voltados para Web, aplicações móveis e sistemas baseados em APIs. 2021 não foi considerado nessa interpretação por ser um ano fração neste mapeamento, sendo que a busca nas bases ocorreu no final de janeiro.

Fazendo um paralelo entre a distribuição de estudos apresentada na Figura 11 e os dados apresentados pela indústria de software voltada para a área de segurança, observa-se que o crescente interesse da academia tem sido motivado pelo aumento de ciber-ataques reportados pela indústria nos últimos anos. No que diz respeito aos ciber-ataques, o relatório da organização *Internet Security Center* apontou que no primeiro semestre de 2018 foram identificados 795.000 novos softwares maliciosos todos os dias Ma *et al.* (2019). Ainda sobre softwares maliciosos, em 2018 surgiu o *Ransomware* chamado *WannaCry*[1], que, no primeiro dia de "lançamento", foi capaz de infectar mais de 120.000 dispositivos em 99 países. Em relação aos ataques de DDoS, o relatório da organização NETSCOUT apontou que a quantidade de ataques deferidos no período de 2017 a 2018 aumentou 26% (CALVERT; KHOSHGOFTAAR, 2019). Em relatório apresentado pela empresa Kaspersky, o número de ataques no período de 2018 a 2019 aumentou 18% (Benzaïd; Boukhalfa; Taleb, 2020), sendo que a quantidade de ataques pode chegar a 15.4 milhões até o ano de 2023 segundo relatório apresentado pela empresa Cisco (ZHOU *et al.*, 2020). Em ataques deferidos contra aplicações web, o relatório *"Symantec Internet Security*

[1] Software malicioso que criptografa arquivos em computadores e exige um regaste para obter uma chave que desbloqueie os arquivos.

Figura 11 – Distribuição dos estudos por ano de publicação (2018 a 2020)



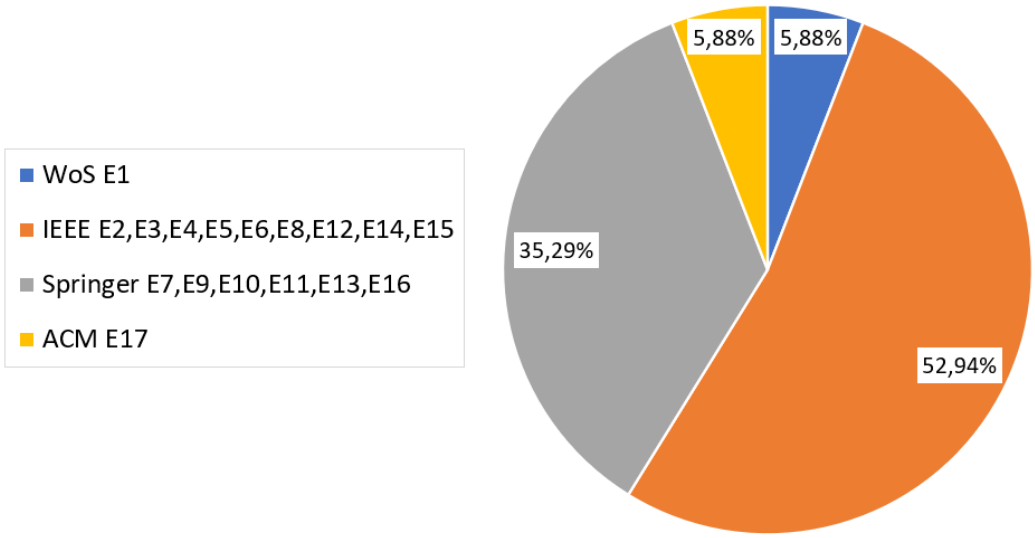
Ano	Estudos	Total
2018	E1,E5,E14	3
2019	E2,E4,E10,E12,E13,E16	6
2020	E3,E6,E7,E8,E9,E11,E15,E17	8

Fonte: Autoria própria

Report" apontou que em 1.3 milhões de ataques coletados em 2018, 10% das URLs (do inglês, *Uniform Resource Locator*) alvos eram maliciosas (YANG; ZHOU; CUI, 2020). Os dados apresentados por esses relatórios nos permite estabelecer algumas evidências sobre a área de pesquisa deste trabalho. A primeira e mais concreta está relacionada ao importante aumento dos ataques em relação aos números de estudos encontradas na literatura. De acordo com as análises deste SMS, a academia não tem tratado esse assunto como solução completa do ponto de vista de engenharia, fornecendo uma solução que possa atuar de maneira proativa e reativa em relação aos ciber-ataques. Notou-se, que esse assunto tem sido tratado de maneira isolada, sendo propostas soluções que aplicam algoritmos específicos para solução de problemas extremamente particulares.

Já o segundo cenário visa apresentar a distribuição de estudos em cada base de pesquisa, como ilustra a Figura 12. Essa figura revela que as bases IEEE e Springer somam ≈88,23% dos estudos selecionados para este mapeamento. Outro ponto importante a ser observado é a ausência das bases de pesquisa Science Direct e Scopus nessa distribuição. Embora, as bases supracitadas tenham retornado 59 e 258 estudos respectivamente, ao final deste SMS nenhum estudo foi selecionado. No que diz respeito ao conteúdo, pode-se dizer que a Science Direct é uma base que indexa em sua maioria periódicos e artigos estendidos de conferência em uma fração menor. Em contrapartida, a Scopus é uma base que indexa várias outras bases, não sendo tão perceptível essa proporção entre artigos publicados em periódicos e conferências. Além disso, vale ressaltar

Figura 12 – Distribuição dos estudos por base de pesquisa

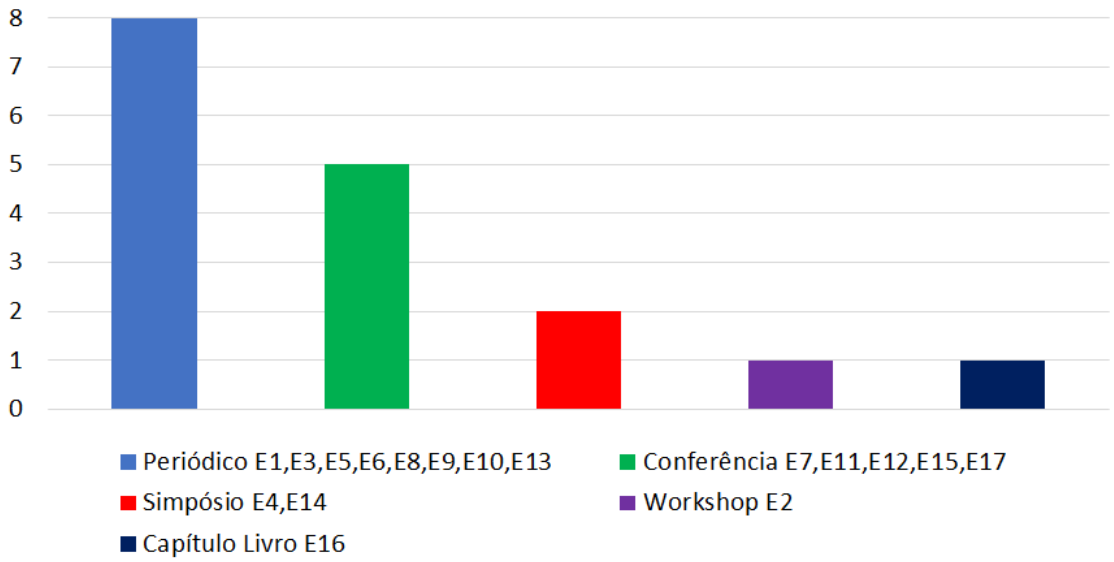


Fonte: Autoria própria

que os artigos encontrados não estão nela disponíveis, mas sim nas bases de origem.

Complementando o segundo cenário de informação, a Figura 13 ilustra a distribuição dos estudos contidos neste SMS quanto à origem de publicação, caracterizando o terceiro cenário de um trabalho dessa natureza. Para isso, os estudos deste SMS (veja Apêndice B) foram classificados em cinco tipos, a saber: periódicos, conferências, simpósios, *workshops* e capítulos de livros.

Figura 13 – Distribuição dos estudos por tipo de publicação



Fonte: Autoria própria

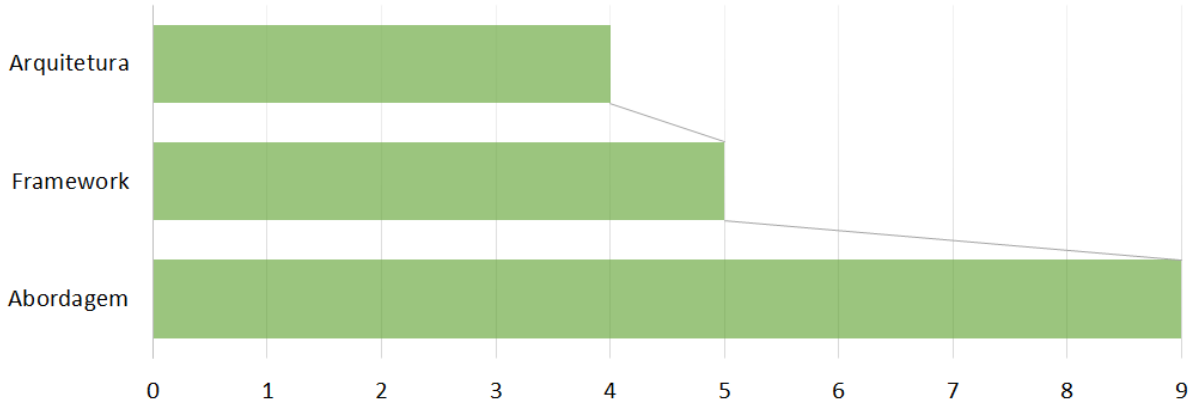
Analisando os três cenários apresentados, observa-se que oito estudos foram publicados em periódicos e nove nas demais origens de publicação. Embora próximos, essa diferença pode

reforçar a evidência apresentada para o primeiro cenário de informação, ou seja, a academia não tem tratado esse tema de pesquisa como uma solução de autoproteção de maneira mais ampla e completa voltada para aplicações Web, móveis e baseados em APIs. Em paralelo, observou-se também que os esforços dos interessados foram direcionados para a proposição e/ou melhoria de algoritmos novos ou já existentes na literatura. Portanto, de acordo com as informações reportadas nesta seção, essa evidência pode estar relacionada a complexidade inerente a esse tipo de solução e, ao mesmo tempo, a natureza perecível desse tipo de solução em relação ao surgimento constante de ciber-ataques. A seguir são reportadas as respostas para cada QP deste SMS (veja Apêndice A).

QP 1: Quais são as soluções de autoproteção que foram elaboradas para apoiar o desenvolvimento de SApps?

Ao final da etapa de extração dos 17 estudos, três tipos de soluções foram identificados, conforme ilustrado na Figura 14. Abordagem com nove estudos foi o tipo de solução mais recorrente. *Framework* e arquitetura completam os demais tipos de solução com cinco e quatro estudos, respectivamente. Em relação aos números, nota-se que a soma dos valores excede o total de estudos do SMS. Isso ocorre porque o estudo E12 foi classificado como uma solução que representa uma abordagem voltada para Arquitetura, cobrindo dois tipos de solução. A seguir, uma descrição sobre cada tipo de solução é reportada.

Figura 14 – Distribuição dos estudos pelo tipo de solução de autoproteção



Fonte: Autoria própria

1. **Abordagem**. Estudos classificados neste tipo de solução podem estar associados a diferentes estágios de desenvolvimento. Na visão de Lakatos (2003), o estudo pode estar em alto nível de abstração, caracterizando que o tema de estudo esteja em estágios iniciais de pesquisa (LAKATOS, 2003). Em contrapartida, pela definição de SEVOCAB (2021), o

estudo pode apresentar uma solução que necessita sistematizar o desenvolvimento por meio de diretrizes, práticas de engenharia e processos automatizados (por exemplo, bibliotecas, ferramentas e *frameworks*).

2. **Arquitetura**. Embora o termo "arquitetura de software" seja subjetivo e conflitante entre alguns autores, nesta trabalho foi adotada a visão de Fowler (2019). Nesta visão, arquitetura de software é uma compreensão compartilhada do *design* de um sistema pelos desenvolvedores especialista em um projeto, sendo que na compreensão compartilhada estão contidos os principais componentes do sistema e como os mesmos interagem entre si. Portanto, estudos classificados neste tipo de solução apresentam evidências sobre os componentes e conectores arquiteturais voltados para autoproteção de sistemas.

3. ***Framework***. De acordo com Fayad (1999), *framework* é um software quase completo direcionado a um domínio de aplicação que podem ser facilmente customizado e reutilizado para aplicações específicas que atendam a requisitos de negócio. Diante do exposto, estudos classificados neste tipo de soluções fornecem autoproteção de sistemas por meio de um software que pode ser integrado e customizado ao sistema alvo em desenvolvimento.

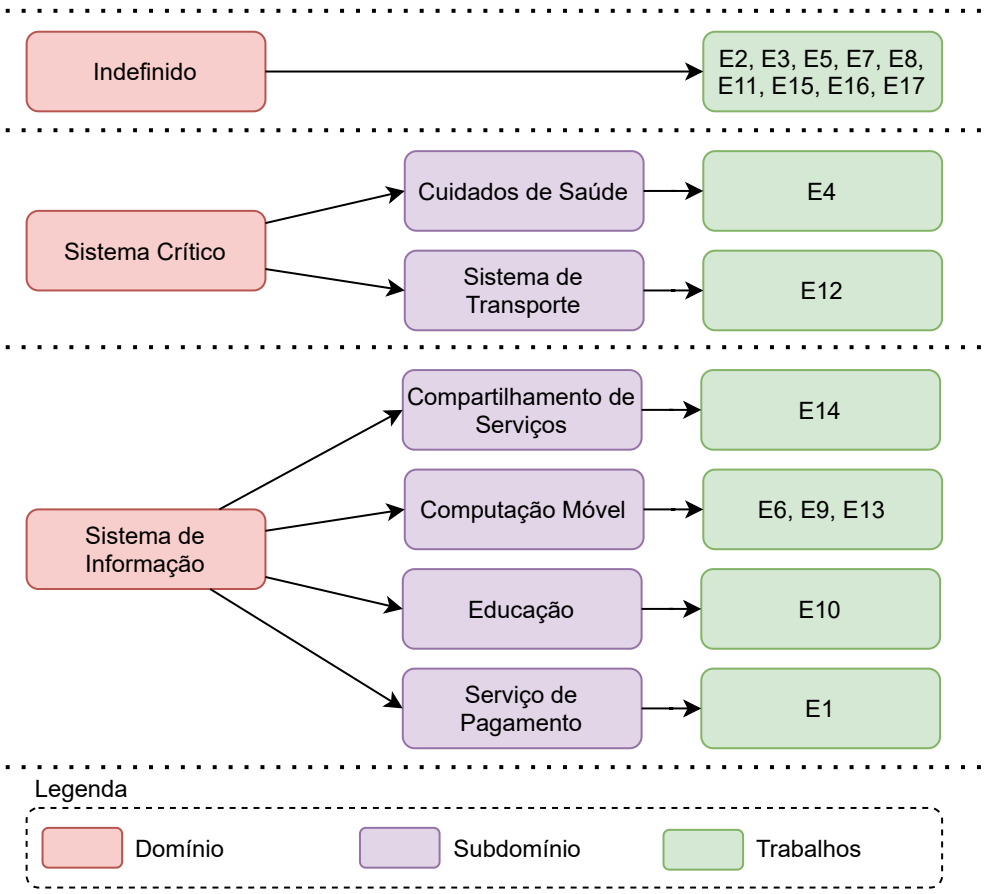
A Figura 14 revela que 9 estudos empregam o uso de abordagem (E2, E6, E7, E9, E10, E11, E12, E13, e E14) e representam ≈52,94% dos 17 estudos extraídos, enquanto aqueles que empregam *framework* (E1, E4, E8, E15 e 16) e arquitetura (E3, E5, E12 e E17) representam ≈29,41% e ≈23,52%, respectivamente. Fazendo um paralelo entre as soluções do tipo "Abordagem" e o número total de estudos deste SMS, observa-se que existe um número baixo de soluções que podem ser consideradas mais sólidas do ponto de vista de engenharia de sistemas. Embora tais iniciativas sejam importantes para mostrar a atuação da comunidade científica neste tema, a quantidade de estudos existente permite inferir que este tema encontra-se em estágio inicial de desenvolvimento e necessita de maior investigação e aplicabilidade em cenários reais de execução.

QP 2: Quais são os domínios de aplicação que se beneficiaram por ter soluções de autoproteção para SApps?:

Esta questão de pesquisa teve como meta identificar os domínios e subdomínios de aplicações que implementaram soluções de autoproteção. Deste modo, conforme ilustrado na Figura 15, três domínios de aplicações foram identificados, a saber: (i) Sistemas de informação; (ii) Sistemas críticos; e (iii) Indefinido. O primeiro é composto por quatro subdomínios (ou seja, "Serviço de Pagamento", "Educação", "Computação Móvel" e "Compartilhamento de Serviços") e representa ≈35,29% dos estudos. Basicamente, aplicações desse domínio oferecem funcionalidades que lidam com a manipulação de dados de seus usuários. O segundo representa

≈11,76% dos estudos, sendo composto por dois subdomínios (ou seja, "Cuidados de Saúde" e "Sistema de Transporte"). As aplicações desse domínio são consideradas críticas por lidarem com a integridade física dos usuários finais e/ou de seu ambiente de execução. Por fim, o terceiro e mais presente nos estudos deste SMS com ≈52,94% (E2, E3, E5, E7, E8, E11, E15, E16, E17), representam aplicações em que não foi possível identificar um domínio de aplicação. Esses números reforçam a evidência apontada na QP 1 sobre os tipos de soluções voltadas apenas para lidar com autoproteção como um assunto específico. A seguir, definições sobre cada subdomínio identificado são reportadas:

Figura 15 – Distribuição dos estudo por domínios e subdomínios



Fonte: Autoria própria

1. **Compartilhamento de Serviços.** Estão relacionadas a este subdomínio aplicações que compartilham algum tipo de serviço por meio da Internet como, por exemplo, serviços de compartilhamento de arquivos e multimídia. Existe apenas um estudo incluído neste subdomínio (E14).

2. **Computação Móvel.** Aplicações deste subdomínio envolvem dispositivos capazes de efetuar algum nível de processamento em que normalmente utilizam tecnologia de rede sem fio para comunicação. Essas aplicações são relacionadas a dispositivos como

smartphones, *smartwatches* e IoT. Três estudos foram incluídos neste subdomínio (E6, E9, E13).

3. **Cuidados de Saúde.** São aplicações que permitem o monitoramento da saúde de pacientes ou que possam auxiliar em alguns procedimentos clínicos. Apenas um estudo está contido neste subdomínio (E4).

4. **Educação.** Este subdomínio inclui aplicações que processam e disponibilizam serviços relacionados ao ensino e educação em geral. Apenas um estudo está contido neste subdomínio (E10).

5. **Serviço de Pagamento.** São aplicações que tratam de transações financeiras entre pessoas e/ou instituições. Um estudo está incluído neste subdomínio (E1).

6. **Sistema de Transporte.** As aplicações contidas neste subdomínio estão relacionadas ao controle e/ou gerenciamento de trânsito de automóveis ou de veículos autônomos. Existe uma aplicação incluída neste subdomínio (E12).

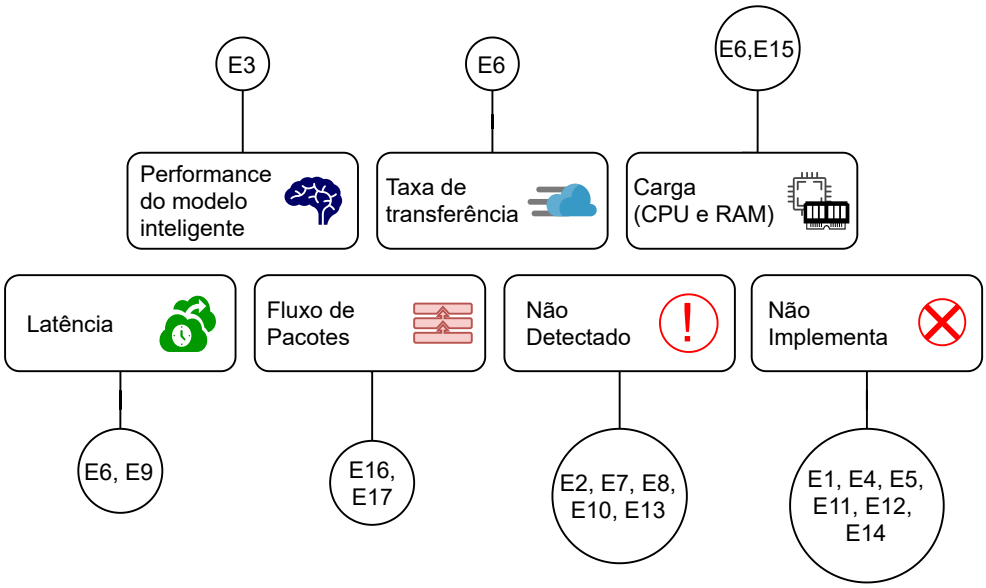
Embora esse tema de pesquisa tenha mostrado ser de grande importância e interesse por parte da academia e indústria, a combinação das informações dessas duas questões de pesquisa mostra que os avanços nessas áreas são preliminares do ponto de vista de engenharia de sistemas. As iniciativas identificadas são voltadas, em geral, para soluções menores que abrangem a concepção e/ou melhoria de algoritmos e validações dos mesmos em relação a um tipo de ataque específico. Nesse sentido, os dados apresentados para esta questão de pesquisa são relevantes para mostrar quais tipos de aplicações podem requerer soluções de autoproteção no contexto deste SMS.

QP 3: Quais são os atributos de qualidade utilizados no projeto de soluções de autoproteção para SApps?

O objetivo desta questão de pesquisa foi extrair evidências sobre o uso de atributos de QoS (do inglês, *Quality of Service*) nas soluções de autoproteção deste mapeamento. Em outras palavras, esses atributos fornecem métricas e parâmetros aos SApps para que seja possível detectar, por exemplo, comportamentos maliciosos ou ameaças/vulnerabilidades em tempo de execução. Como pode ser observado na Figura 16, em ≈35,29% dos estudos não mencionam utilizar qualquer tipo de métrica que possa ser associada a um atributo de qualidade, fazendo com que os mesmos fossem categorizados como "Não implementa". Em ≈29,41% dos estudos foi atribuído a categoria "Não detectado", pois esses estudos utilizam algum tipo de métrica sem que fosse possível identificar seu cenário de atuação. No entanto, em linhas gerais, essas métricas atuam em paralelo à solução de autoproteção como forma de complementar a detecção de

ameaças/vulnerabilidades. Métricas relacionadas a "Latência", "Estado de Protocolos" e "Carga (CPU e RAM) representam ≈11,76% dos estudos para cada categoria. Por fim, "Performance do modelo inteligente" e "Taxa de transferência" com ≈5,88% dos estudos para cada categoria fecham as métricas utilizadas nas soluções de autoproteção. A seguir são apresentadas as definições de cada categoria.

Figura 16 – Distribuição dos estudos por atributos de QoS



Fonte: Autoria própria

1. **Performance do modelo inteligente.** Esta categoria utiliza um tipo de métrica que está relacionada aos ataques que buscam perturbar o modelo inteligente de detecção de vulnerabilidades/ataques. Portanto, a performance do modelo inteligente deve ser monitorada a fim de detectar possíveis ataques de perturbações. Apenas o estudo **E3** foi classificado nessa categoria.

2. **Taxa de Transferência.** Nesta categoria a abordagem de autoproteção monitora o meio de transmissão de dados, sendo este meio cabeado ou sem fio. As ameaças/vulnerabilidades são identificadas quando ocorrem anomalias na velocidade de transferência. Apenas o estudo **E6** foi incluído nesta categoria.

3. **Carga (CPU e RAM).** Esta categoria está relacionada ao uso de recursos de processamento como uso de CPU e memória RAM. Em geral, um processo de monitoramento de uma solução de autoproteção tenta identificar altas cargas de processamento/memória que fogem do comportamento normal para alertar outros sistemas que eventuais ataques podem estar ocorrendo. Os estudos **E6** e **E15** foram incluídos nessa categoria.

4. **Latência.** De acordo com SEVOCAB (2021), o termo latência está relacionado ao intervalo de tempo entre o instante em que uma unidade de controle de instrução emite uma chamada de dados e o instante que a transferência de dados é iniciada. Neste sentido, os estudos incluídos nesta categoria possuem abordagens de autoproteção que monitoram a latência na transmissão de dados a fim de detectar comportamentos anômalos ou ataques. Foram incluídos nesta categoria os estudos **E6** e **E9**.

5. **Fluxo de Pacotes.** Os estudos incluídos nesta categoria analisam fluxos de pacotes de rede com auxílio de modelos de aprendizado de máquina, onde diferentes tipos de ameaças/vulnerabilidades podem ser detectados. Por permitir interpretações distintas em relação aos possíveis atributos de qualidade que podem estar presentes nos estudos **E16** e **E17**, essa categoria representa de modo genérico para lidar com os mesmos, fazendo que o fluxo possa ser implementado em formato de algum tipo de métrica capaz de lidar com o atributo de interesse de uma aplicação.

6. **Não Detectado.** Os estudos incluídos nesta categoria reportam evidências que indicam o uso de atributos de qualidade (métricas) que foram utilizados para auxiliar na identificação de ameaças/vulnerabilidades pelas soluções de autoproteção. Como a maioria dos estudos é voltada para explorar técnicas de ciência de dados ou algoritmos inteligentes ao invés de engenharia de sistemas, não foi possível resgatar os nomes e/os significados dos atributos de qualidade como citado na literatura. Foram incluídos nesta categoria os estudos **E2**, **E7**, **E8**, **E10** e **E13**.

7. **Não Implementa.** Foram incluídos nesta categoria os estudos em que suas soluções de autoproteção não utilizam atributos de qualidade (ou algum tipo de métrica) para identificar ameaças/vulnerabilidades. Foram incluídos nessa categoria os estudos **E1**, **E4**, **E5**, **E11**, **E12** e **E14**.

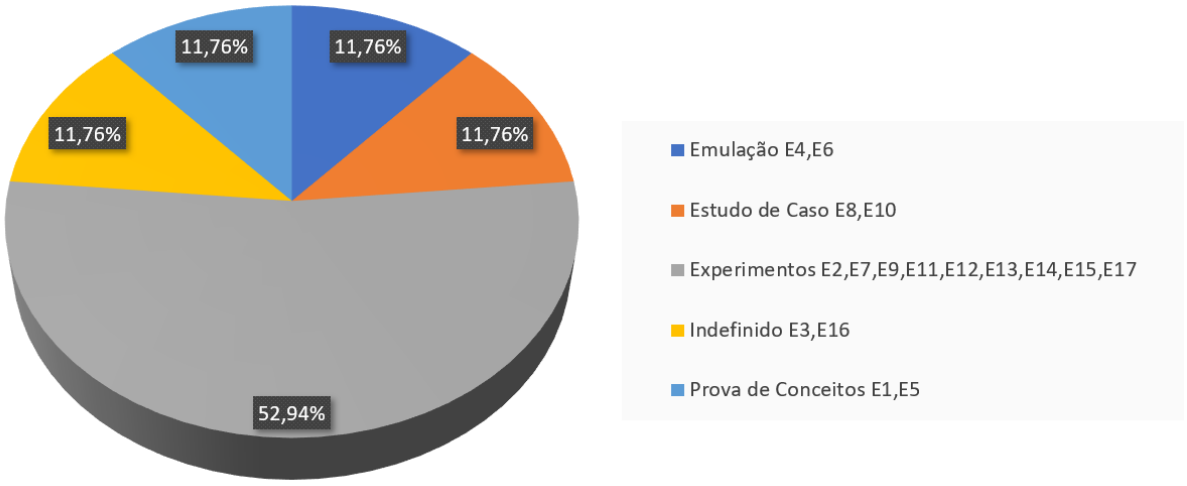
De acordo com os dados extraídos nessa questão de pesquisa, ≈64,70% dos estudos não implementam atributos de qualidade ou não possuem evidências concretas sobre a utilização dos mesmos para apoiar a atividade de autoproteção. Em contrapartida, o estudo **E6** deve ser destacado por lidar com três atributos de qualidade simultaneamente, a saber: “Taxa de transferência”, “Carga (CPU e RAM)” e “Latência”. De maneira complementar, ≈35,30% dos trabalhos reportaram aplicar atributos de qualidade como uma alternativa complementar às soluções propostas. Em geral, os atributos de qualidade (ou métricas) podem ser utilizados em abordagens reativas para detecção de ameaças/vulnerabilidades, que entram em ação quando as abordagens proativas falham ou não foram implementadas. Neste sentido, por meio da análise das métricas por eles implementadas/disponibilizadas é possível detectar comportamentos anormais, que podem ser interpretados como ameaças/vulnerabilidades. Portanto, pode-se dizer que a

maioria dos trabalhos extraídos nesse mapeamento possuem abordagem de autoproteção passíveis de falhas em ataques do tipo *zero day*, situações em que abordagens proativas são passíveis de falhas.

QP 4: Como as soluções de autoproteção para SApps têm sido avaliadas?

Esta questão de pesquisa apresenta um panorama sobre como os estudos extraídos avaliaram as soluções de autoproteção. De acordo com a Figura 17, apenas dois estudos (ou seja, ≈11,76%) não foi possível identificar algum tipo de avaliação nas soluções apresentadas. Em contrapartida, 15 estudos (≈88,24%) apresentaram algum tipo de avaliação, sendo distribuída da seguinte maneira: ≈52,94% do estudos avaliaram suas soluções por meio de experimentos (E2, E7, E9, E11, E12, E13, E14, E15, E17); ≈11,76% por meio de emulação (E4, E6); ≈11,76% utilizaram prova de conceito (E1, E5); e, por fim, ≈11,76% dos estudos utilizaram estudo de caso (E8, E10). A seguir, são apresentadas definições sobre cada um dos tipos de avaliação:

Figura 17 – Distribuição dos estudos pelo tipo de avaliação



Fonte: Autoria própria

1. **Experimento.** Este tipo de avaliação é realizado com base em um conjunto de dados pré-estabelecidos para determinados cenários e em um protocolo de execução bem definido. Assim, características do objeto alvo podem ser avaliadas e os resultados obtidos são analisados para que parâmetros em relação aos comportamentos esperados e inesperados sejam extraídos (PASSINI *et al.*, 2021).

2. **Emulação.** De acordo com SEVOCAB (2021), este termo representa o desenvolvimento de um modelo que aceite/imite as entradas e saídas que um determinado sistema ou ambiente poderia fornecer. No contexto deste SMS, sistemas foram projetados para emular situações

de ataques para que o comportamento de uma solução de autoproteção pudesse ter sido avaliada.

3. **Estudo de Caso.** Este tipo de avaliação é utilizado em pesquisas exploratórias em ambientes onde o pesquisador não tem controle sobre as variáveis, sendo que o foco é direcionado aos eventos contemporâneos. Portanto, podem ser empregados na validação de resultados de pesquisas para entender, explicar ou demonstrar as capacidades de uma nova técnica, método, ferramenta, processo, tecnologia ou estrutura organizacional (PERRY; SIM; EASTERBROOK, 2004).

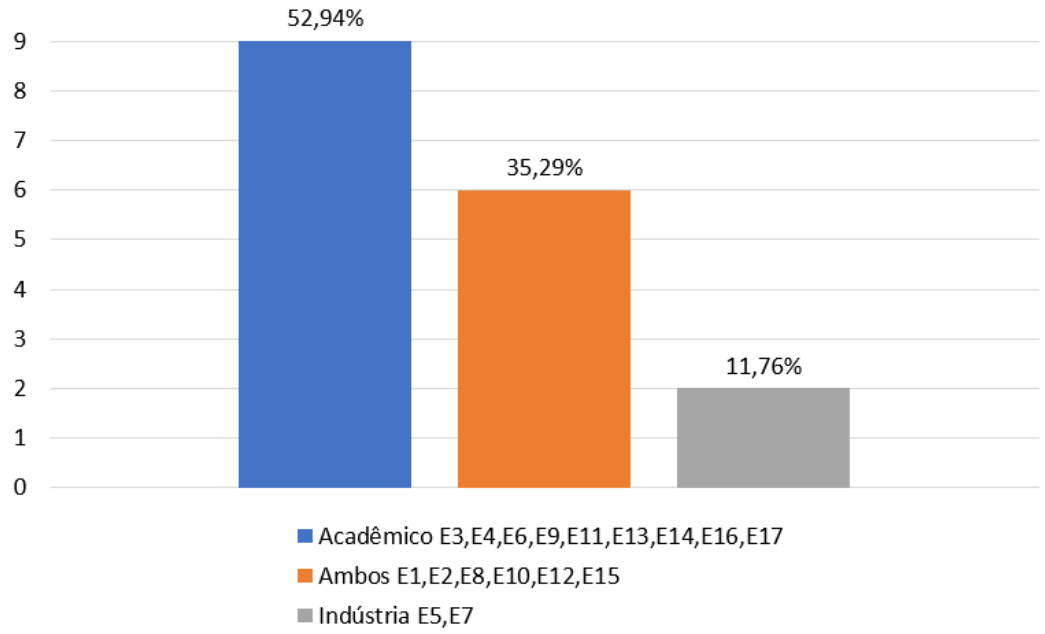
4. **Prova de Conceito.** De acordo com Gregg, Kulkarni e Vinzé (2001), quando um conceito de sistema representa mudanças substanciais em sistemas existentes, um modelo ou protótipo é necessário como uma prova de conceito para demonstrar a viabilidade de construção de um novo sistema ou aprimoramento dos existentes. Corroborando com esta visão, esse termo "*Proof of Concepts*" é definido em SEVOCAB (2021) como o desenvolvimento de um modelo ou protótipo para demonstrar a viabilidade de uma ideia ou tecnologia.

QP 4.1: Quais são as evidências que motivaram a adoção de soluções de autoproteção em SApps?

Nesta questão de pesquisa foram extraídas evidências sobre a aplicabilidade das soluções de autoproteção na indústria, academia ou ambas. Visando estabelecer um panorama das soluções de autoproteção, os estudos foram classificados como acadêmico, que são aqueles que apresentaram abordagens teóricas e/ou experimentais, sendo avaliados por meio de experimentos, emulações ou comparações, entre outros. Em contrapartida, os estudos classificados como pertencentes à indústria são aqueles que apresentaram evidências de domínio e escopo bem definidos e tiveram como resultados *frameworks* e/ou protótipos para emprego em cenários reais de aplicação. Já os estudos de natureza mista são aqueles que apresentam intersecção entre a academia e indústria. A Figura 18 ilustra a distribuição dos estudos em função da classificação supracitada.

De acordo com os resultados apresentados na Figura 18, ≈52,94% dos estudos possuem aplicação estritamente acadêmica (E3, E4, E6, E9, E11, E13, E14, E16, E17), ≈35,29% em ambos (E1, E2, E8, E10, E12, E15) e ≈11,76% são voltados para a indústria (E5, E7). Estudos classificados como ambos podem ser caracterizados como aqueles que tiveram o desenvolvimento na academia e algum tipo de aplicabilidade em cenários reais de execução na indústria. Os números revelam que as iniciativas existentes estão desequilibradas em distribuição, havendo uma maioria significativa de estudos de origem acadêmica. Esses dados corroboram com

Figura 18 – Distribuição dos estudos entre academia e indústria



Fonte: Autoria própria

nossas evidências anteriores sobre o estágio de desenvolvimento das soluções encontradas neste mapeamento.

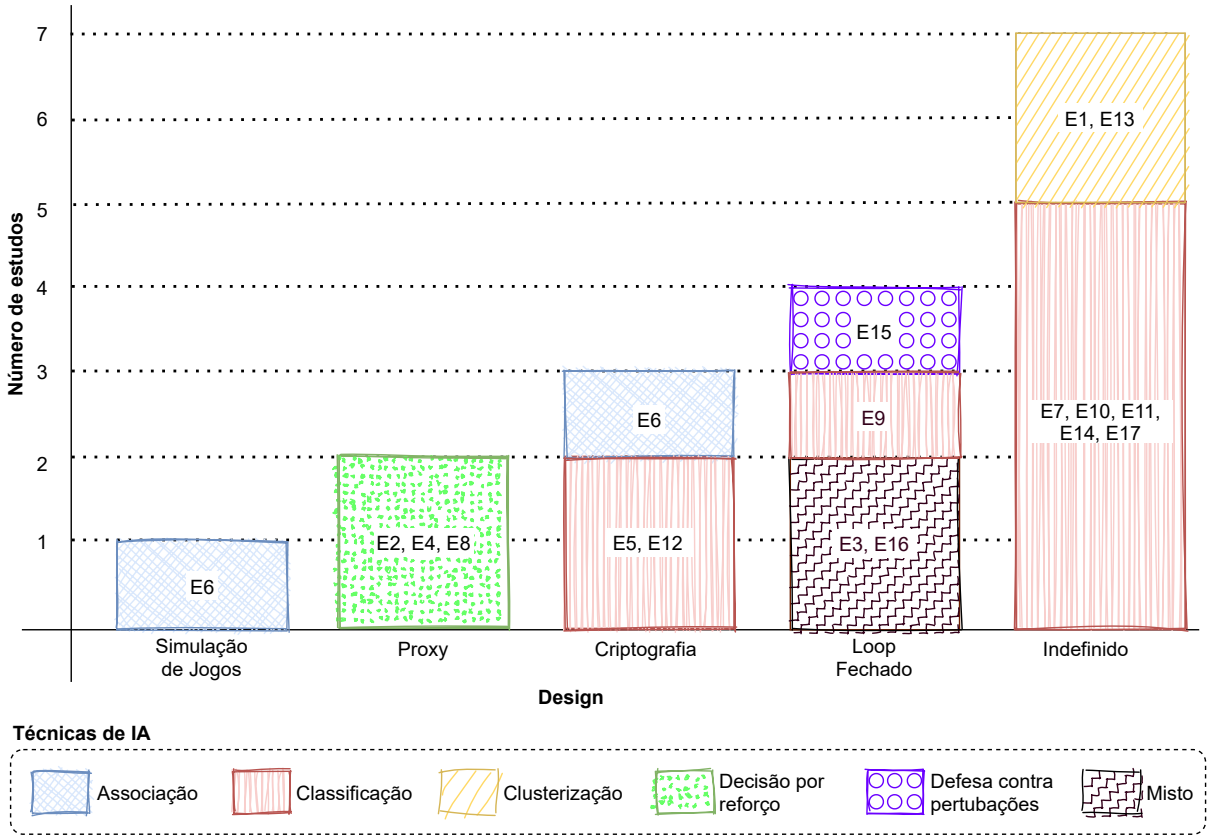
QP 5: Como as soluções de autoproteção para SApps têm sido projetadas?

Esta questão tem como objetivo reunir evidências sobre o projeto das soluções de autoproteção. A Figura 19 ilustra a combinação entre os recursos utilizados no *design* e as técnicas de inteligência artificial (IA) empregada no projeto de tais soluções. Visando mapear como esses recursos e técnicas têm sido utilizados, os mesmos foram agrupados em duas categorias chamadas "*Design*" e "Técnicas de IA". Também é possível observar nessa figura que um conjunto de recursos/técnicas foi identificado para cada categoria, mostrando como as mesmas podem ser utilizadas de maneira isolada ou combinadas para atender aos interesses de uma solução. A seguir, definições sobre categorias e técnicas são apresentadas:

Design. Esta categoria reúne os recursos/técnicas utilizados(as) no *design* das soluções de autoproteção. Como forma de uniformizar a descrição desta questão, os recursos/técnicas serão referenciados(as) deste ponto em diante apenas como item identificado desta categoria. A seguir, uma descrição de cada item é apresentada:

1. **Simulação de Jogos.** Estudos que reportam *designs* baseados em técnicas relacionadas a teoria de jogos, como por exemplo, "Equilibro de Nash" foram incluídos neste item. De modo geral, a simulação de jogos permite modelar comportamentos de agentes defensores

Figura 19 – Distribuição dos estudos entre os recursos utilizados no *design* e as técnicas de IA



Fonte: Autoria própria

e maliciosos a fim de disponibilizar uma abordagem de autoproteção. Foi identificado apenas um estudo neste item (E6).

2. ***Proxy.*** Neste item encontram-se estudos baseados em estruturas *proxy*, que são baseadas em componentes intermediários implementados entre a comunicação de dois sistemas ou entidades. Deste modo é possível interceptar as mensagens trocadas, detectar possíveis ataques e/ou anomalias e tomar alguma decisão visando realizar ações protetivas para o sistema. Foram identificados dois estudos para este item (E2 e E8).

3. **Criptografia.** É um recurso aplicado ao *design* baseado em métodos criptográficos para transmissão dos dados em uma rede como modo de evitar ataques do tipo "*man in the middle*". Esse tipo de ataque pode interceptar a comunicação entre sistemas e adulterar seu conteúdo a fim de comprometer a integridade e/ou a confidencialidade dos dados. Três estudos foram incluídos neste item (E5, E6 e E12).

4. ***Loop* Fechado.** Estudos classificados neste item devem ter fases bem definidas no processo de autoproteção. Geralmente essas fases são compostas por coleta e tratamento dos dados, planejamento de ações protetivas, e execução das ações planejadas. Além disso, as ações

executadas devem ter mínima interferência humana, o que caracteriza um ciclo fechado de ações. Foram identificados quatro estudos para este item (E3, E9, E15, E16).

5. **Indefinido**. Foram classificados como indefinidos os estudos que não apresentam abordagens de *design* voltados a engenharia de sistemas. Em geral, os estudos aqui classificados apresentam apenas técnicas voltadas a área de ciência de dados e/ou algoritmos inteligentes. Foram incluídos neste item sete estudos (E1, E7, E10, E11, E13, E14 e E17).

Técnicas de IA. Esta categoria engloba um conjunto de técnicas que simulam o comportamento inteligente para prover autoproteção por meio de decisões automatizadas, que podem ocorrer, por exemplo, por meio de técnicas de classificação, associação, clusterização ou uma combinação de ambas. A seguir, uma breve definição sobre cada uma das técnicas é apresentada:

1. **Associação.** Por meio de um conjunto de regras é possível determinar a ocorrência de elementos/ações em meio a um conjunto de dados, permitindo assim encontrar relacionamentos entre conjuntos de exemplos (BUCZAK; GUVEN, 2016). Apenas um estudo foi identificado para esta técnica (E6).

2. **Classificação.** Por meio de um modelo de aprendizado criado a partir de um conjunto de dados rotulados, um conjunto de atributos é extraído a partir de comportamentos e/ou ações do ambiente/sistema alvo e é classificado em uma determinada classe de dados. Esta técnica foi identificada em nove dos estudos selecionados (WITTEN *et al.*, 2011) (E1, E5, E7, E9, E10, E11, E12, E14 e E17).

3. **Clusterização.** Visa segmentar um conjunto de dados por meio de suas características em comum, sem a supervisão de um especialista humano para rotular os dados previamente (NORVIG; RUSSELL, 2021). Foram identificados dois estudos baseados nesta técnica (E1 e E13).

4. **Decisões por reforço.** Técnica empregada em situações com o objetivo de aprender uma sequência de ações ou tarefas necessárias para cumprir um objetivo específico. Por meio de recompensas ou punições, um agente pode aprender como tomar decisões em ambientes desconhecidos (NORVIG; RUSSELL, 2021). Foram identificados três estudos (E2, E4 e E8) baseados nesta técnica.

5. **Defesa contra pertubações.** Sistemas de autoproteção que utilizam métodos inteligentes para detectar ameaças ou anomalias podem ter seus modelos de aprendizado perturbados por ações maliciosas, fazendo com que os modelos de aprendizado gerados deixem de detectar corretamente ataques maliciosos (Benzaïd; Boukhalfa; Taleb, 2020). As soluções de defesa atuam contra tais pertubações tentando impedir que os modelos de aprendizado

tenham influências negativas que possam prejudicar seu funcionamento. Foi identificado apenas um estudo baseado nesta técnica (E15).

6. **Misto.** São soluções de autoproteção que fazem uso de duas ou mais técnicas inteligentes para identificar ameaças/vulnerabilidades de segurança. Foram identificados dois estudos (E3 e E16).

Conforme ilustrado na Figura 19, ≈42,18% dos trabalhos não apresentaram evidências sobre os recursos de *design* utilizados no projeto de suas soluções, ou seja, esses estudos reportam apenas melhorias e evoluções em algoritmos e/ou técnicas voltada para a área de ciência de dados. Desses estudos, apenas dois utilizaram clusterização (E1 e E13) e cinco empregaram classificação (E7, E10, E11, E14 e E17) como técnicas de IA no *design* de suas soluções. Criptografia (E6, E5 e E12), simulação de jogos (E6) e *proxy* (E2, E4 e E8) estão presentes em ≈35,29% dos estudos deste mapeamento como recursos utilizados no *design* de suas soluções. No que diz respeito à combinação de recursos e técnicas, pode-se dizer que o *design* está alinhado com técnicas de IA para permitir que ameaças/vulnerabilidades sejam detectadas de maneira automática. Em paralelo, observa-se que essa combinação tem resultado em capacidade de autonomia indesejada, ou seja, as soluções ainda necessitam de intervenções humanas para funcionamento no ambiente de execução.

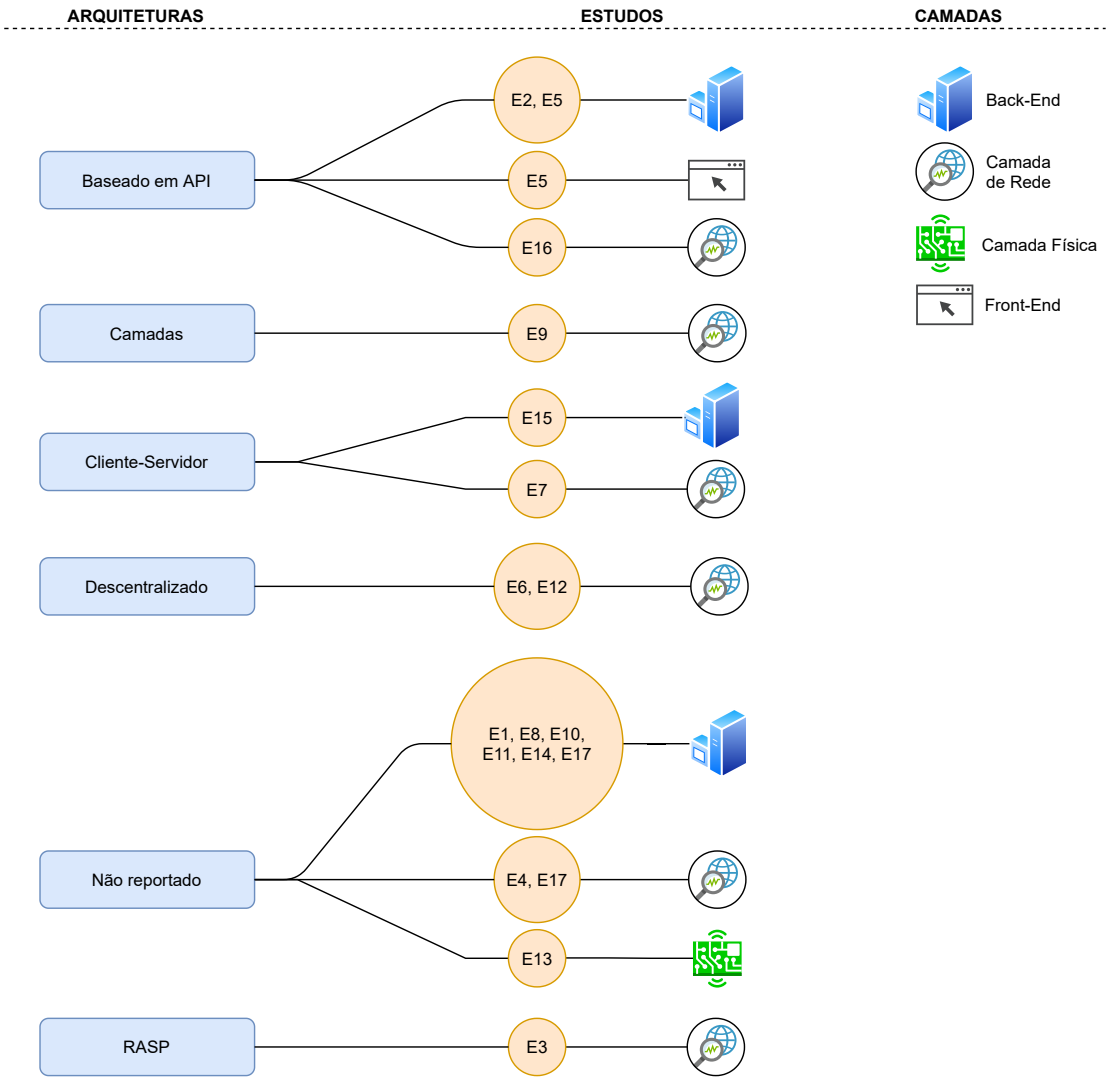
Em um outro ponto de vista, nota-se que ≈23,52% dos estudos foram utilizados *loop* fechado como recurso de *design* aliado com as técnicas de IA (E3, E9, E15 e E16). Essa combinação tem revelado maior indicativo de autonomia das soluções por dois motivos: (i) organização da solução em fases bem definidas pela utilização do *loop*; e (ii) combinação das técnicas de IA para lidar com ataques em tempo de execução. Dentre os estudos supracitados, apenas o estudo E15 é voltado a camada de aplicação, sendo seu foco de defesa ataques DDoS. Diante do exposto, pode-se notar que os dados apresentados nesta questão evidenciam que a abordagem de autoproteção proposta por este trabalho de mestrado possui inovação significativa em relação aos trabalhos existentes na literatura do ponto de vista de engenharia de solução e possibilidade de escalabilidade/flexibilização para lidar com ameaças/vulnerabilidades de sistemas.

QP 5.1: Como as soluções de autoproteção para SApps têm sido organizadas?

O propósito desta questão de pesquisa foi reunir evidências sobre a organização das soluções de autoproteção do ponto de vista arquitetural. Para isso, procurou-se entender quais estilos arquiteturais têm sido utilizados e em quais camadas a solução proposta visa atuar. Conforme ilustrado na Figura 20, ≈47,05% dos estudos (E1, E4, E8, E10, E11, E13, E14 e E17) não foi possível identificar evidências concretas sobre a organização arquitetural da solução, sendo estes classificados como “Não reportado” (E1, E4, E8, E10, E11, E13, E14 e E17). Em ≈17,65% dos estudos (E2, E5 e E16) foi identificado o modelo “Baseado em API”. “Cliente

Servidor" (E7 e E15) e "Distribuído" (E6 e E12) foram os modelos identificados em ≈11,76% dos estudos cada. Por fim, as arquiteturas "RASP" (E3) e "Camadas" (E9) estão presentes em ≈6,88% dos estudos cada. A seguir são apresentadas as definições de cada modelo de arquitetura:

Figura 20 – Distribuição dos estudos entre as arquiteturais e camadas



Fonte: Autoria própria

1. **Baseado em API.** API é uma interface composta por um conjunto de regras e protocolos padronizados que definem como processos ou componentes trocam mensagens. Em outras palavras, as APIs permitem a integração de diferentes serviços sem que seja necessário conhecer a implementação interna de cada componente (IBM, 2020).

2. **Camadas.** Este tipo de arquitetura é aplicado em *designs* que a comunicação entre os elementos de um sistema ocorre de modo unidirecional, sendo que as camadas são empilhadas uma sobre as outras. Deste modo, uma camada pode comunicar-se apenas com a que esteja localizada na fronteira. Logo, cada camada tem capacidade de abstrair

complexidades de implementação para a camada localizada em seu entorno (BASS; KAZMAN; CLEMENTS, 2012).

3. **Cliente-Servidor.** É um modelo arquitetural que tem sido utilizado na autoproteção de sistemas em dois processos, sendo um cliente e outro servidor. O processo servidor é responsável por fornecer algum serviço, que é requisitado e consumido por processos clientes. Os processos podem residir em um mesmo *host* ou diferentes, que são interligados por um meio de uma rede de comunicação (TANENBAUM, 2007).

4. **Descentralizado.** Este modelo de arquitetura de software possui fortes semelhanças com a arquitetura cliente-servidor, diferenciando-se pelo fato dos processos poderem assumir papel de servidor e cliente. Portanto, todos os processos são capazes de fornecer os mesmos serviços e a comunicação entre os componentes ocorrem ponto-a-ponto (TANENBAUM, 2007).

5. **Não reportado.** Os estudos incluídos nessa categoria não reportam evidências concretas sobre os modelos arquiteturais utilizados nas soluções de autoproteção.

6. **RASP.** A arquitetura RASP (do inglês, *Runtime Application Self-Protection*), permite que sistemas sejam protegidos contra ciberataques em tempo de execução. Portanto, são aplicados conceitos de autoproteção que permitem a autoconfiguração do software com pouco ou nenhuma intervenção humana. Para isso, são definidos pontos de monitoramento (PM) em locais específicos da aplicação que permitem detectar entradas ou comportamentos suspeitos (CISAR; CISAR, 2016).

No que diz respeito às camadas dos SApps, onde foram aplicadas técnicas *design*, a Figura 20 ilustra que em ≈52,94% do estudos foi reportada a camada "*Back-End*" (E1, E2, E5, E8, E10, E11, E14, E15 e E17), ≈47,06% dos estudos foi reportada "Camada de Rede" (E3, E4, E6, E7, E9, E12, E16 e E17) e as camadas "*Front-End*" (E5) e "Camada Física" (E13) foram reportadas em ≈6,88% dos estudos para cada camada. Na sequência são apresentadas as definições de cada camada:

1. ***Back-end.*** É a camada responsável pelo acesso aos dados e execução das regras de negócios de uma aplicação. Nesta questão de pesquisa, serão consideradas as soluções que protegem aplicações, aplicativos e sistemas operacionais. Foram detectados nove estudos que lidam com camada em um SApp.

2. **Camada de Rede.** Estão incluídas nesta camada as SApps que possuem algum nível de proteção na comunicação entre os componentes da aplicação por meio de uma rede, sendo que essa comunicação pode estar associada a qualquer camada do modelo OSI (do inglês,

Open Systems Interconnection model). Foram identificados oito estudos que abordam esta camada.

3. **Camada Física.** Nesta camada, as soluções de autoproteção tem como objetivo promover proteção em dispositivos de hardware como sensores e dispositivos de comunicação. Foi identificado apenas um estudo que aborda esta camada.

4. ***Front-end*.** Conhecida também como camada de apresentação, é por onde usuários podem interagir com um domínio de sistema, através de interfaces que podem variar em nevadores Web, *smartphones*, *smartwatches*, entre outros. Foi identificado apenas um estudo que aborda esta camada.

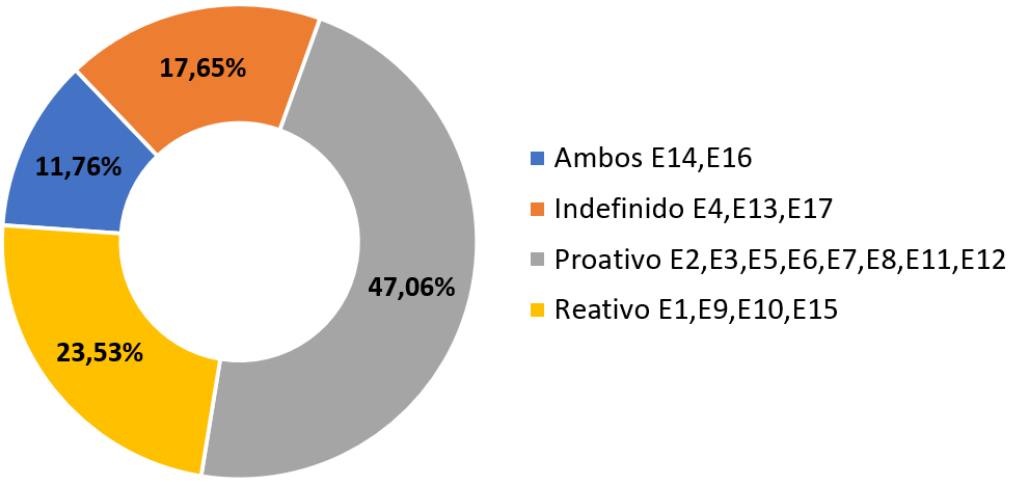
Os estudos que reportaram aplicar autoproteção nas camadas "Camada de Rede" e "Camada Física" somam ≈52,94% dos estudos selecionados (E3, E4, E6, E7, E9, E12, E13, E16 e E17). Esses estudos direcionam seus pontos de monitoramento para o perímetro das aplicações de domínio, onde têm capacidade de detectar apenas ameaças/vulnerabilidades que exploram o fluxo de dados de entrada. Neste sentido, esse tipo de organização apresenta deficiência na detecção ameaças/vulnerabilidades que exploram falhas de *design* e/ou lógica de negócio. Além disso, ≈35,29% do estudos (E1, E8, E10, E11, E14 e E17) que reportaram trabalhar na camada "*Back-End*", mas não foi possível identificar claramente qual modelo de arquitetura utilizado na solução. Ao final, os dados apontam que apenas ≈17,65% dos estudos (E2, E5 e 15) extraídos têm a capacidade de detectar anomalias e/ou ataques ocasionados por falhas de *design* e/ou lógica de negócio. Portanto, os dados evidenciam e corroboram com as Questões de Pesquisa 1 e 2 no sentido que os estudos que apresentam abordagens de autoproteção possuem carência em técnicas de engenharia de sistemas.

QP 6: Quais estratégias foram usadas no projeto de soluções de autoproteção para SApps?

Nesta questão de pesquisa são apresentadas as estratégias de autoproteção identificadas neste SMS, como ilustra a Figura 21. Como pode ser observado, oito estudos (≈47,05%) exploram somente o aspecto "Proativo' e quatro estudos (≈23,53%) apenas o "Reativo". Dois estudos foram classificados como "Ambos" porque exploram os aspectos proativo e reativo simultaneamente em uma solução de autoproteção. Por fim, em três estudos (≈17,64%) foram classificados como "Indefinido", pois não foi possível encontrar algum tipo de evidência concreta que permitisse classificá-los em uma dessas estratégias. A seguir, definições para cada estratégia é apresentada:

1. **Ambos.** Esta estratégia representa soluções de autoproteção capazes de empregar ações proativas para as ações maliciosas já conhecidas e, ao mesmo tempo, ações reativas para as

Figura 21 – Distribuição dos estudos por estratégias de autoproteção



Fonte: Autoria própria

que forem desconhecidas. Foram identificados dois estudos nessa categoria (E14 e E16).

2. **Indefinido.** Representam estudos que não tornaram explicitas e/ou implícitas as suas estratégias de autoproteção. Quatro estudos foram identificados para essa categoria (E4, E13 e E17).

3. **Proativo.** Esta estratégia representa soluções de autoproteção com a capacidade de antecipar um comportamento anômalo em tempo de execução. Tais comportamentos geralmente são recorrentes e já conhecidos pelo sistema de autoproteção. Foram identificados oito estudos nessa categoria (E2, E3, E5, E6, E7, E8, E11 e E12).

4. **Reativo.** Neste tipo de estratégia ocorrem ataques desconhecidos no qual a solução de autoproteção não é capaz de detectar de maneira antecipada. Em geral, esses ataques podem ser classificados em dois cenários: (i) conhecidos na literatura/indústria e não implementado na solução; e (ii) ataques conhecidos como “dia de zero”. Com base nos cenários expostos, uma solução de autoproteção deve ser capaz de detectar ações maliciosas com base em comportamentos fora do comum que ocorrem na aplicação de domínio. Quatro estudos foram identificados nessa categoria (E1, E9, E10 e E15).

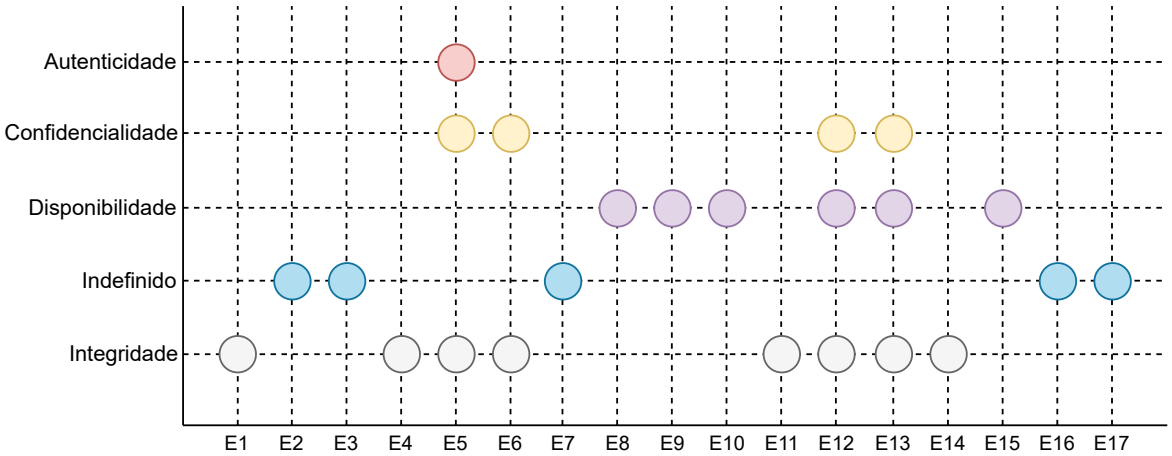
Como pode ser observado, apenas dois estudos deste SMS propuseram soluções de autoproteção de maneira proativa e reativa. O estudo E14 apresenta uma abordagem capaz de complementar o conhecimento adquirido em tempo de *design* com conhecimento descoberto em tempo de execução para identificar comportamentos futuros que necessitem de adaptação de segurança. Já no estudo E16 foram abordados ataques ativos de camada cruzada em redes sem fio. A proposta deste estudo é baseada em dois módulos, sendo uma para detecção de

ataques e outro para mitigação de ações maliciosas desconhecidas. Por fim, vale destacar que ≈88,23% dos estudos abordaram as estratégias de maneira isolada (≈70,58%) ou indefinida (≈17,64%), o que corrobora com nossas constatações nas questões anteriores sobre a cobertura das soluções de autoproteção. Conforme evidenciado neste capítulo, as soluções de autoproteção são consideradas perecíveis quanto aos tipos de ataques/ameaças por elas suportados. Tal constatação está relacionada ao surgimento constante de novos ataques/ameças e, ao mesmo tempo, as adversidades de projeto dessas soluções para que as mesmas possam ser escaláveis e/ou incrementais. Embora algumas iniciativas possam ser identificadas neste SMS, fica evidente que a área de pesquisa explorada neste mapeamento necessita de avanços significativos para promover soluções mais sólidas e robustas que possam ser aplicadas em ambientes reais de execução.

QP 7: Quais são as dimensões da segurança das soluções de autoproteção abordadas em SApps?

Esta questão de pesquisa teve por objetivo reunir evidências sobre as dimensões de segurança empregadas no desenvolvimento das SApps. Na Figura 22 é apresentado um mapa que ilustra a distribuição dos estudos em cada dimensão. “Integridade”, “Disponibilidade” e “Confidencialidade” foram as dimensões de segurança mais recorrentes neste SMS com ≈47,05%, ≈35,29% e ≈23,52%, respectivamente. Com apenas ≈5,88% dos estudos, “Autenticidade” foi a quarta dimensão identificada. Por fim, “Indefinido”, com ≈29,41%, representa o conjunto de estudos em que não foi possível identificar uma dimensão de segurança. A seguir, uma breve definição sobre cada dimensão de segurança é apresentada:

Figura 22 – Distribuição dos estudos pela dimensão de segurança



Fonte: Autoria própria

1. **Autenticidade.** Nesta dimensão de segurança, o mecanismo de proteção busca validar

o acesso de um ator a determinados recursos do sistema. Os atores podem ser pessoas, dispositivos ou sistemas de informação.

2. **Confidencialidade.** Por meio de um conjunto de controles de segurança, os SApps tentam proteger dados de acessos não autorizados, sendo que esses dados podem estar em trânsito, em processamento, ou armazenados em banco de dados.

3. **Disponibilidade.** As ações protetivas visam impedir que ações maliciosas tornem um sistema protegido indisponível ou com acesso limitado.

4. **Indefinido.** Estão incluídos nesta categoria estudos que apresentam dimensões genéricas de segurança, onde os mecanismos de proteção não lidam com assuntos específicos como os demais apresentados nesta questão.

5. **Integridade.** Os mecanismos de proteção objetivam impedir que dados sejam modificados ou substituídos por meio de ações que ocorrem geralmente por meio de acessos não autorizados. Assim como na confidencialidade, os dados também podem estar em trânsito, em processamento, ou armazenados em banco de dados.

Ao avaliar os dados extraídos por essa QP, pode-se observar que ≈47,05% (E1, E4, E8, E9, E10, E11, E14 e E15) dos estudos extraídos abordam apenas uma dimensão de segurança, ≈17,65% (E5, E12 e E13) dos estudos abordam 3 dimensões e ≈6,88% (E6) dos estudos abordam duas dimensões. Portanto, os números evidenciam que a maioria dos estudos reportam soluções de autoproteção capazes de lidar com apenas um grupo específico de ameaças. Esse cenário de distribuição sugere que a integração de dimensões de segurança tem se revelado uma alternativa viável para defender uma aplicação de domínio.

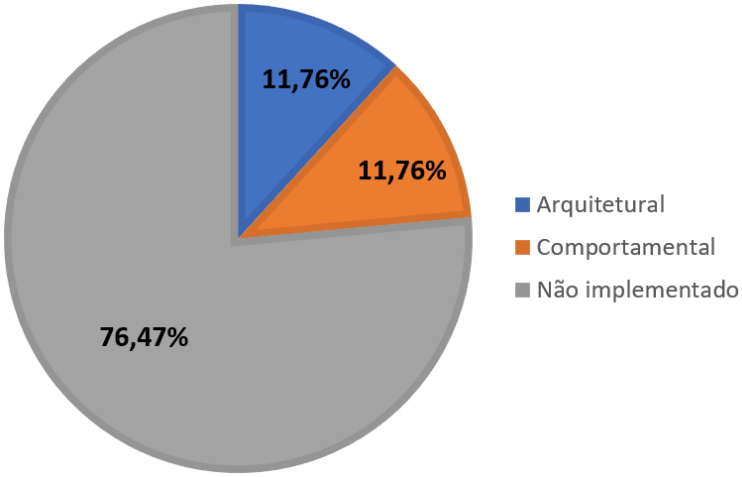
Outro ponto que deve ser destacado é que em ≈29,41% (E2, E3, E7, E16 e E17) dos estudos não são reportadas claramente quais são as dimensões de segurança em que a solução de autoproteção atua. Portanto, esses estudos não têm condição de apontar quais são os pontos de monitoramento da SApp, que são componentes essenciais para este tipo de solução. Os números reportados para esta questão reforçam a evidência principal deste mapeamento sobre a cobertura das soluções de autoproteção e o potencial de escalabilidade das mesmas, quando outros interesses de proteção necessitam ser incorporados.

QP 8: Quais são as modificações realizadas pelas soluções de autoproteção em SApps?

Esta questão reúne evidências sobre os tipos de modificações realizadas pelas soluções de proteção nas aplicações para impedir e/ou mitigar ações maliciosas. As soluções de autoproteção identificadas neste SMS foram projetadas com base em dois tipos de modificação, a saber:

(i) “Comportamental” com ≈11,76% dos estudos (E9, E15); e (ii) “Arquitetural” com ≈11,76% dos estudos (E5, E8). Nos demais estudos (E1, E2, E3, E4, E6, E7, E10, E11, E12, E13, E14, E16 e E17) não foram encontradas evidências de modificação na aplicação (SApps) pelos mecanismos de defesa das soluções encontradas, que representam ≈76,47% dos estudos. Na Figura 23 é ilustrada a distribuição dos tipos de modificação empregados nas soluções de autoproteção para SApps. A seguir são apresentadas as definições para tipo de modificação:

Figura 23 – Distribuição dos estudos pelos tipos de modificação



Fonte: Autoria própria

1. **Arquitetural.** Este tipo de modificação representa adição, remoção ou substituição de módulos/componentes de uma aplicação pela solução de autoproteção. Além disso, novas composições de componentes/módulos também podem ser formadas por meio da modificação de seus conectores.

2. **Comportamental.** De modo análogo ao tipo de modificação arquitetural, as operações ocorrem em nível de granularidade menor. Dessa forma, funcionalidades, ações ou métodos podem ser adicionados, removidos ou substituídos para que as novas necessidades de autoproteção sejam atendidas.

3. **Não implementado.** São estudos que não aplicam modificações no sistema protegido quando ataques ou anomalias são identificadas.

Como pode-se observar, a maioria dos estudos (≈76,47%) não reportam como as soluções de autoproteção realizam modificações em tempo de execução nas SApps. Esses números sugerem que as soluções de autoproteção apresentadas por esses estudos podem se tornar indisponíveis por indeterminados períodos de tempo após algum tipo de ataque por serem fortemente dependentes de intervenções humanas após a etapa de detecção do mesmo. Por outro

lado, ≈23,53% desses estudos reportaram ter utilizado modificações em tempo de execução, permitindo que as aplicações de domínio aumentem sua capacidade de resiliência de maneira autônoma em situações que apresentem comportamentos anômalos.

QP 9: Quais são as técnicas de aprendizado abordadas nas soluções de autoproteção para SApps?

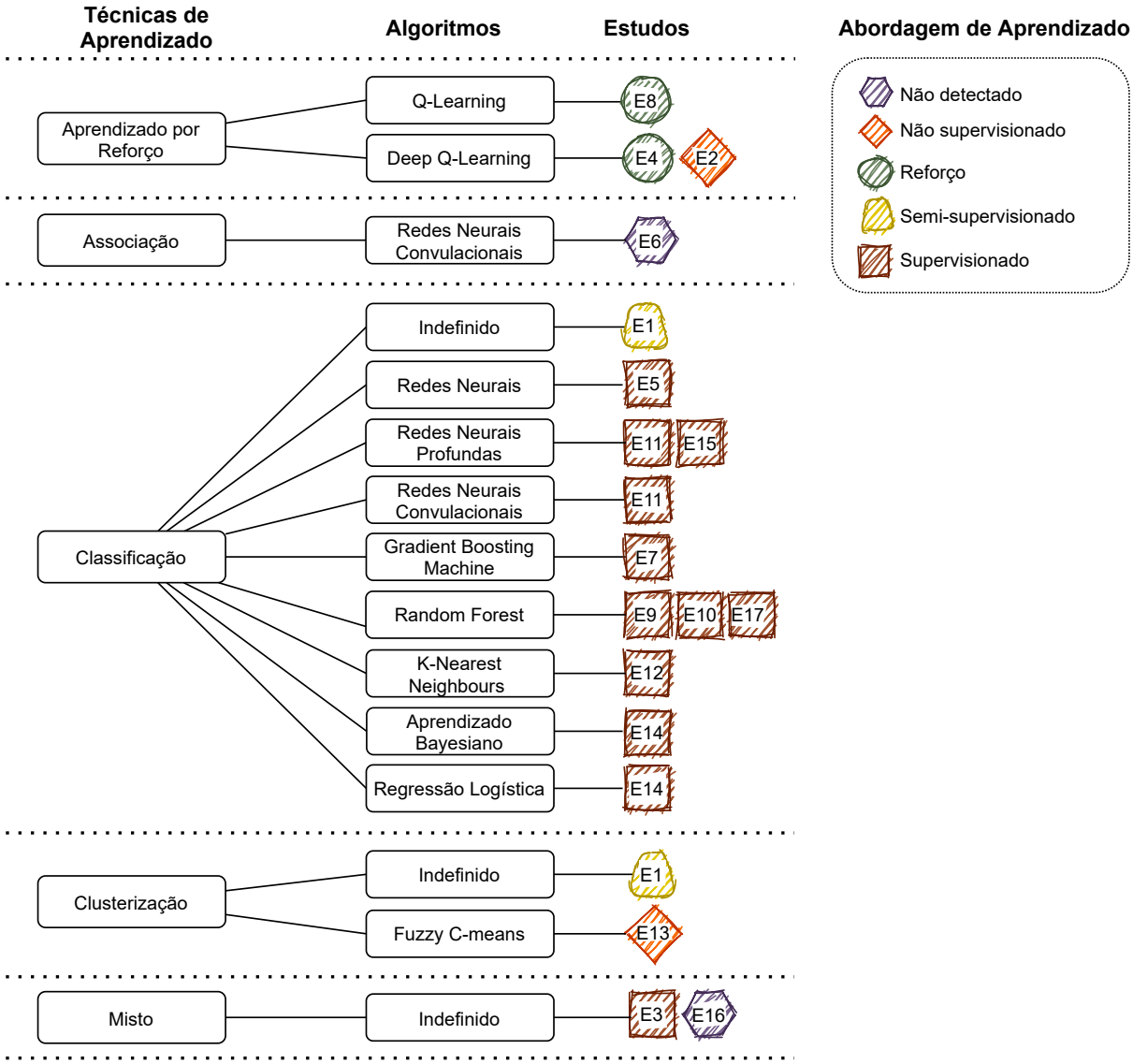
São apresentadas nesta questão as técnicas inteligentes e os respectivos algoritmos utilizados nas soluções de autoproteção para SApps, conforme ilustra a Figura 24. "Classificação", com 12 ocorrências, foi a técnica de aprendizado mais recorrente dos estudos deste SMS. "Aprendizado por Reforço", "Clusterização", "Associação", com, respectivamente, três, duas e uma ocorrências, fecham os números das técnicas identificadas neste SMS. Apenas dois estudos foram classificados como "Misto" por apresentarem mais de uma técnica de aprendizado.

Em relação aos algoritmos de aprendizado, pode ser observado que "Redes Neurais" e suas variantes "Redes Neurais Profundas", "Redes Convulacionais", além do algoritmo *"Deep Q-Learning"* que utiliza redes neurais profundas como suporte, estão presentes em ≈41,18% dos estudos (E2, E4, E5, E6, E11 e E15), sendo o algoritmo mais recorrente neste mapeamento. Na sequência aparece o algoritmo *"Random Forest"* com uso em ≈17,65% dos estudos, e o restante dos algoritmos sendo aplicados em ≈5,88% dos estudos cada.

Sobre as abordagens de aprendizado reportadas pelos estudos, cinco categorias foram identificadas: (i) "Supervisionado" em 10 estudos (E3, E5, E7, E9, E10, E11, E12, E14, E15 e E17); (ii) "Não supervisionado" em dois estudos (E2 e E13); (iii) "Semi-supervisionado em um estudo (E1); (iv) "Aprendizado por reforço" em dois estudos (E4 e E8); e (v) "Não Detectado" em dois estudos (E6 e E16). A descrição de cada abordagem de aprendizado pode ser visualizada na Seção 2.3.1.

Em especial, observa-se que o estudo E14 reportou ter utilizado os algoritmos "Aprendizado Bayesiano" e "Regressão Logística", porém isso não significa que a abordagem de autoproteção utiliza os dois algoritmos ao mesmo tempo. Os algoritmos foram avaliados individualmente na solução proposta e se mostraram eficientes, embora os mesmos apresentem características distintas. Conforme reportado no estudo E14, no início das avaliações quando a quantidade de registros que representam ações não maliciosas são maiores do que os registros que representam ações maliciosas, o algoritmo bayesiano tende a não identificar alguns ataques. Em contrapartida, o algoritmo de regressão logística tem capacidade de identificar mais ataques, porém, com maior taxa de falsos positivos. Com o passar do tempo, a quantidade de registros que representam ações maliciosas aumentam e o algoritmo bayesiano é capaz de detectar uma taxa maior de ataques.

Figura 24 – Distribuição dos estudos pelas técnicas de aprendizado



Fonte: Autoria própria

O estudo E1 também merece atenção especial por reportar o uso das abordagens de "Classificação" e "Clusterização". Logo, o objetivo da solução de autoproteção é identificar ataques a sistemas de *Fintechs* (acrônimo do inglês, *Financial Technology*). Portanto, a técnica de aprendizado "Clusterização" é aplicada para descobrir novas técnicas de ataque agrupando transações semelhantes, ou seja, são identificados nos rótulos. Assim, após as transações serem rotuladas, a técnica de aprendizado "Classificação" é aplicada para classificar novas transações. Neste sentido, de acordo com (MOHRI *et al.*, 2012), este estudo pode ser classificado como uma abordagem semi-supervisionada. Este tipo de técnica é aplicado em situações em que a quantidade de dados rotulados é menor em relação aos dados não rotulados.

3.3 Discussão

Nesta seção é apresentada uma análise comparativa entre os estudos que compõem esse SMS, sendo que são avaliados três aspectos chaves relacionados a este projeto de mestrado, a saber:

Aspecto 1 – *Apresenta abordagens de engenharia de sistemas?* Para responder a este aspecto foram observadas as questões QP 5 e QP 5.1. O aspecto é considerado atendido caso a QP 5 ou QP 5.1 apresente, respectivamente, alguma abordagem concreta de *design* ou de arquitetura;

Aspecto 2 – *Possui capacidade de monitoramento interno?* Este aspecto está relacionado aos pontos de monitoramento que coletam dados responsáveis pela identificação de ataques e/ou comportamentos anômalos em uma SApps. Este aspecto visa estabelecer se a solução de autoproteção tem capacidade de monitoramento dos componentes internos ou periféricos da aplicação de domínio. Para responder este aspecto foi avaliada a QP 5.1. Assim, pode-se dizer que o aspecto é atendido quando a solução de autoproteção é aplicada na camada "*Back-End*" ou "*Font-End*" de uma SApp;

Aspecto 3 – *Possui capacidade de resiliência a ataques?* Este aspecto visa identificar a capacidade de resiliência em relação aos ciberataques. De acordo com Björck *et al.* (2015), resiliência cibernética pode ser definida como a capacidade que um sistema computacional tem em entregar resultados esperados e se recuperar após ciberataques. Portanto, neste aspecto foram avaliados três questões de pesquisa, as quais foram organizadas da seguinte maneira: **[Aspecto 3.1]** na QP 6 foi verificado se os estudos têm capacidade proativa e reativa contra ciberataques; **[Aspecto 3.2]** na QP 8 foi observado se as soluções de autoproteção podem modificar as aplicações de domínio quando ameaças/vulnerabilidades são identificadas; e, por fim, **[Aspecto 3.3]** na QP 7 foi observado se o estudo aponta claramente a dimensão de segurança tratada pela solução de autoproteção.

A Figura 25 ilustra a distribuição dos aspectos em relação aos estudos deste mapeamento. Fazendo uma análise dessa distribuição, apenas dois estudos têm capacidade de defesa proativa e reativa (E14 e 16) – Aspecto 3.1. No entanto, o estudo E14 não apresenta evidências quanto ao uso das abordagens de engenharia de sistemas. Já o estudo E16 não possui capacidade de monitoramento interno, o que evidencia, por exemplo, pouca capacidade de defesa contra ataques que exploram falhas de *design* ou validação de dados. Além disso, a solução de autoproteção desse estudo não é claro em qual dimensão de segurança atua, além de ser incapaz de modificar-se quando ações maliciosas são detectadas, tornando a aplicação de domínio pouco resiliente a ciberataques e altamente dependente de intervenções humanas após a detecção.

Figura 25 – Comparação entre estudos

	Aspecto 1 - Apresenta abordagens de Engenharia de Sistemas?	Aspecto 2 - Possui capacidade de monitoramento interno?	Aspecto 3 - Possui capacidade de resiliência a ataques?		
Estudos	QP5 e QP5.1 - Trata abordagens de Disign ou Arquitetura?	QP 5.1 - Quais camadas implementam autoproteção?	**Aspecto 3.1** QP 6 - É proativo e reativo?	**Aspecto 3.2** QP 8 - implementa modificações?	**Aspecto 3.3** QP 7 - Trata dimensões de Segurança?
E1	✗	✓	✗	✓	✓
E2	✓	✓	✗	✗	✗
E3	✓	✗	✗	✗	✗
E4	✓	✗	✗	✓	✓
E5	✓	✓	✗	✓	✓
E6	✓	✗	✗	✓	✓
E7	✓	✗	✗	✗	✗
E8	✓	✓	✗	✓	✓
E9	✓	✗	✗	✓	✓
E10	✗	✓	✗	✓	✓
E11	✗	✓	✗	✓	✓
E12	✓	✗	✗	✓	✓
E13	✗	✓	✗	✓	✓
E14	✗	✓	✓	✓	✓
E15	✓	✓	✗	✓	✓
E16	✓	✗	✓	✗	✗
E17	✗	✓	✗	✗	✗

Legenda

✓ Atende ✗ Não Atende

Fonte: Autoria própria

Em relação ao Aspecto 2, pode-se observar que dos dez estudos que atendem positivamente, nove (E1, E2, E5, E8, E10, E11, E13, E15 e E17) não atendem ao Aspecto 3.1. Logo é possível afirmar que embora as soluções de autoproteção desses estudos sejam capazes de detectar ameaças/vulnerabilidades que explorem falhas de *design*, as mesmas apresentam baixa capacidades de resiliência. Algo semelhante ocorre ao observar o Aspecto 1, onde dos onze estudos que atendem positivamente, dez (E2, E3, E4, E5, E6, E7, E8, E9, E12 e E15) estudos não atendem ao Aspecto 3.1. Por fim, pode-se afirmar que nenhum dos estudos deste SMS atendem plenamente aos três aspetos, ou seja, estudos que apresentem abordagens claras de engenharia de sistemas, capacidade de monitoramento interno e capacidade de resiliência a ciberataques. Portanto, pode-se dizer que existe uma demanda de pesquisa reprimida sobre este tema, o que reforça, justifica e valida o tema deste trabalho.

3.4 Considerações finais

O objetivo deste capítulo foi identificar as principais técnicas aplicadas e compreender as deficiências nas soluções de autoproteção presentes nos estudos publicados na literatura, além de validar a viabilidade do tema de pesquisa presente neste trabalho de mestrado.

Na Seção 3.1 foram recuperados da literatura, estudos secundários sobre abordagens de autoproteção voltados a SApps. A partir desse resultado foi possível entender os principais interesses da academia e da indústria de software sobre esse tema de pesquisa, além de identificar os principais termos relacionados ao assunto que foram utilizados nos filtros de busca do SMS abordado na Seção 3.2.

Na sequência, na Seção 3.2 foram apresentados os resultados deste SMS por meio de nove questões de pesquisa. Os resultados extraídos permitiram compreender o atual cenário proposto pelo tema de pesquisa dessa dissertação, a saber: as soluções de autoproteção propostas (QP 1), os domínios de aplicações beneficiados pelas soluções de autoproteção (QP 2), os atributos de qualidade monitorados em aplicações de domínio (QP 3), como foram avaliadas as soluções de autoproteção e quais evidências motivaram a adoção de tais soluções (QP 4 e QP 4.1), como foram projetados as soluções de autoproteção e quais arquiteturas foram adotadas (QP 5 e QP 5.1), quais foram as estratégias de defesa e dimensões de segurança abordadas pelas soluções de autoproteção (QP 6 e QP 7), quais técnicas de modificação são aplicadas em tempo de execução (QP 8), e por fim, quais foram as técnicas de aprendizado abordadas para apoiar as soluções de autoproteção (QP 9).

Por fim, a Seção 3.3 apresentou uma análise comparativa envolvendo os dezessete estudos extraídos do SMS, sendo que com base em três aspectos chaves que compõem este trabalho de mestrado, foi possível apontar quais são as lacunas existente na literatura, e como nosso estudo pode contribuir propondo uma solução de autoproteção para SApps. O capítulo Capítulo 4 apresenta com detalhes a solução de autoproteção proposta neste trabalho.

4 ABORDAGEM DE AUTOPROTEÇÃO PARA SApps

O objetivo deste capítulo é apresentar os detalhes da abordagem de autoproteção para SApps proposta neste projeto de mestrado acadêmico. Em resumo, essa abordagem visa proteger esse tipo de aplicação contra ameaças/vulnerabilidades que possam oferecer riscos à camada de aplicação. Por meio de estratégias proativas e reativas aliadas às técnicas de aprendizagem de máquina, a abordagem proposta é capaz de proteger uma aplicação contra ameaças/vulnerabilidades. Para viabilizar tais estratégias foram adotados o *loop* MAPE-K e algoritmos de aprendizado de máquina, combinados com boas práticas de engenharia de software. Essa combinação permite que a abordagem proposta possa tomar ações protetivas isentas, ou com o mínimo de intervenções humanas, frente às ameaças/vulnerabilidades que podem lidar simultaneamente. Além disso, destaca-se que a organização interna da abordagem em módulos independentes permite que a mesma se torne escalar quanto ao número de ameaças/vulnerabilidades, pois novos modelos de aprendizado podem ser gerados para cenários de ameaças/vulnerabilidades desconhecidos e a ela acoplados ampliando sua capacidade de detecção e mitigação de adversidades. Outro ponto importante a ser destacado é a combinação dos modelos de aprendizado para detecção de ameaças/vulnerabilidades com informações coletadas por uma aplicação em tempo de execução. Tal combinação pode impulsionar a capacidade de identificação de requisições anômalas para uma aplicação, pois os módulos de reconhecimento de ameaças/vulnerabilidades passam a lidar com informações comuns dos mesmos e com comportamentos da aplicação em tempo de execução. Por fim, diferentemente de outras soluções disponíveis na literatura que protegem apenas o perímetro da aplicação, a abordagem proposta neste trabalho é capaz de monitorar também componentes internos da aplicação de domínio em busca de falhas involuntárias geradas na fase de *design*.

4.1 Visão geral da abordagem

A abordagem apresentada neste capítulo é resultado das principais contribuições dos estudos identificados pelo mapeamento reportado no Capítulo 3. Em linhas gerais, a abordagem proposta neste trabalho foi projetada para atender três importantes aspectos identificados na Seção 3.3, sendo que nenhum dos trabalhos apresentados do referido mapeamento atendem esses três aspectos simultaneamente. A seguir, detalhes de cada aspecto são apresentados:

Aspecto 1 – ***Emprego de abordagens de engenharia de sistemas***. A abordagem de autoproteção

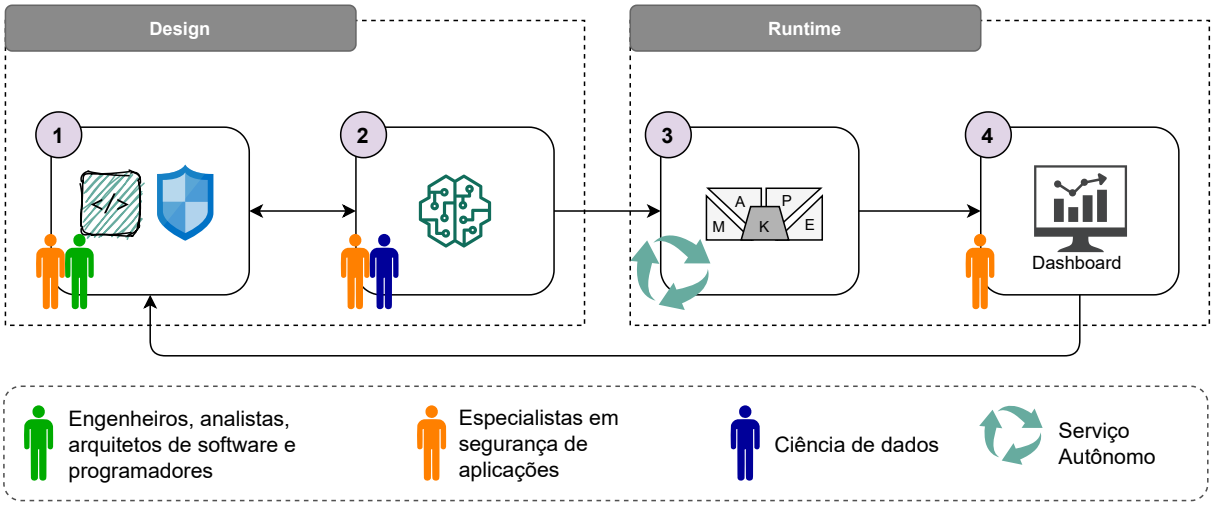
proposta neste trabalho foi organizada em uma topologia de *loop* fechado (MAPE). Essa organização sugere que a abordagem tenha fases bem definidas que são executadas de maneira cíclica. Em síntese, uma aplicação pode ser dividida em duas camadas, sendo uma responsável pela autoproteção e a outra pela aplicação de domínio. Por meio de pontos de monitoramento que podem estar "plugados" em componentes periféricos ou internos da aplicação, ocorre a fase de coleta de dados pelo módulo Monitor (M) da abordagem de autoproteção. A partir dos dados coletados, o módulo de Análise (A) classifica os dados por meio de modelos de aprendizado supervisionado. Em seguida, com base na classificação realizada na fase de análise, o módulo de planejamento (P) determina ações protetivas e/ou adaptativas para serem implementadas na aplicação. Por fim, a partir das ações planejadas, o módulo de execução (E) efetua as ações protetivas e/ou adaptativas por meio de efetores na aplicação de domínio.

Aspecto 2 – *Capacidade de monitoramento interno*. Neste aspecto, a abordagem proposta permite que pontos de monitoramento sejam implantados em componentes periféricos e/ou internos da aplicação. Logo, a abordagem permite que falsos-positivos sejam minimizados consideravelmente pela capacidade de avaliação de comportamento dos componentes internos da aplicação. Essa capacidade é simplificada pelo fato do módulo protetor ser independente da aplicação do domínio, permitindo que ambos os processos estejam hospedados em *hosts* diferentes.

Aspecto 3 – *Capacidade de resiliência*. A abordagem proposta neste trabalho é capaz de tratar ações maliciosas de modo proativo e/ou reativo. Nesse sentido, o primeiro tenta classificar ações maliciosas baseadas em ameaças/vulnerabilidades já conhecidas. A segunda é capaz de atuar em ações maliciosas conhecidas como "dia zero", onde por meio de classificação de comportamentos anômalos da aplicação de domínio (por exemplo, degradação de QoS, interrupção de serviços, acesso indevido, entre outros), ações protetivas são tomadas para mitigar tentativas de ataques. Além disso, por meio de uma abordagem baseada em *logs*, os modelos de classificação podem ser atualizados para que seja possível classificar as atuais ações de dia zero em ataques futuros.

A abordagem apresentada neste capítulo é composta por um projeto de *design*, sendo um processo composto por um conjunto de ações e tarefas, além de algumas ferramentas que permitem automatizar alguns pontos desse processo. A Figura 26 apresenta uma visão geral da abordagem de autoproteção proposta neste trabalho, onde as duas primeiras atividades ocorrem na fase de *design* e as duas últimas na fase de *runtime* de uma aplicação. Basicamente, na **Atividade 1** ocorre o desenvolvimento de uma aplicação, onde participam engenheiros, arquitetos de softwares, programadores e especialistas em segurança de aplicações. Nesta atividade são definidos pontos de monitoramento, que são responsáveis por coletar os dados que serão analisados

Figura 26 – Visão geral da abordagem

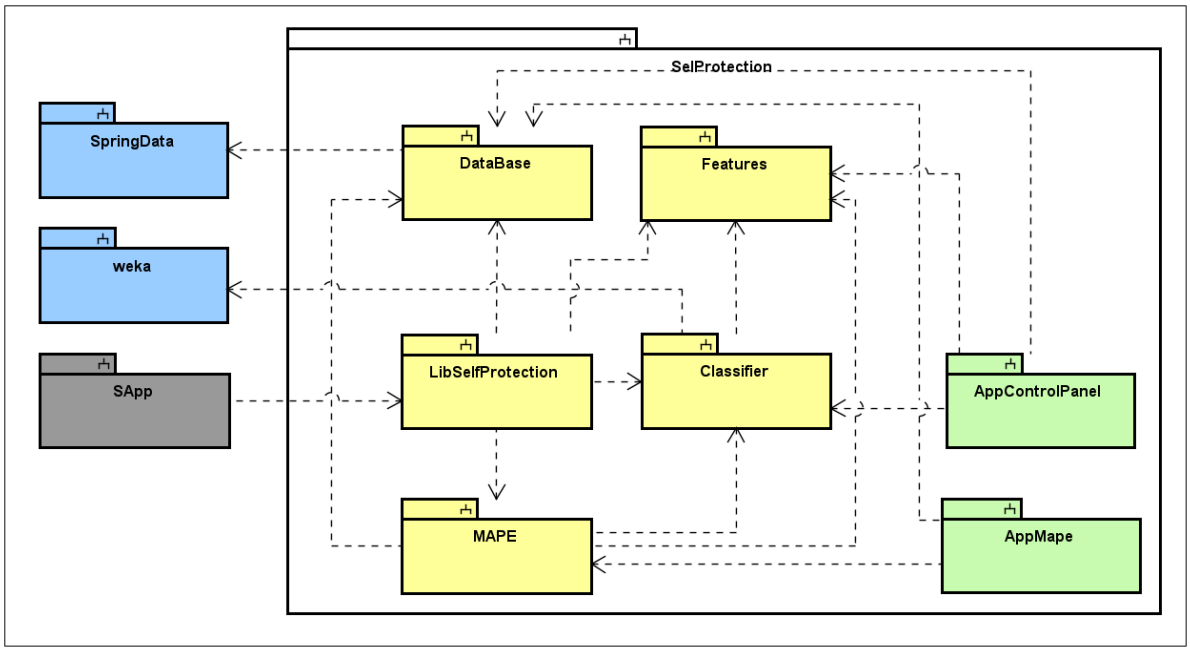


Fonte: Autoria própria

em busca de ações maliciosas e/ou de vulnerabilidades na fase de *runtime*. Na **Atividade 2**, especialistas em segurança e ciência de dados trabalham em atividades de preparação e análise de dados, resultando em um modelo de classificação capaz de detectar vulnerabilidades ou comportamentos maliciosos em uma aplicação. Vale destacar neste momento que as **Atividades 1 e 2** podem ocorrer de maneira cíclica até a definição dos modelos de classificação, pois os resultados da Atividade 2 podem apontar performance insatisfatória em tais modelos e ajustes nos pontos de monitoramento (**Atividade 1**) devem ser realizados para correção dos mesmos. Na **Atividade 3**, a partir dos modelos de classificação gerados na **Atividade 2**, os *loops* MAPE-K são instanciados e executados autonomamente como meio de defesa da aplicação de domínio. Por fim, na **Atividade 4**, por meio de ferramentas de *dashboard* e/ou alertas, especialistas em segurança acompanham o desempenho e performance da **Atividade 3**, realizando intervenções em caso de detecção de algum tipo de anomalia.

A Figura 27 ilustra os módulos que fazem parte da etapa de *design* da abordagem proposta neste trabalho, sendo classificados da seguinte maneira: (i) ***design***, que oferece suporte a fase de desenvolvimento de uma SApp por meio dos módulos `Features`, `Classifier`, `LibSelfProtection`, `MAPE` e `DataBase`; (ii) **automação**, representa o processo automatizado existente na abordagem por meio dos módulos `AppControlPanel` e `AppMape`; (iii) **aplicação**, por meio do módulo SApp, representa a aplicação de domínio a ser protegida; e por fim, (iv) **bibliotecas**, representam as soluções de terceiros que foram utilizadas na abordagem, a saber: módulo `weka`, que oferece suporte para o processo de mineração, classificação de dados e recomendação de informações, e módulo *SpringData*, que oferece suporte para gerenciamento e persistência de dados. No que diz respeito aos módulos supracitados, vale destacar que os mesmos foram implementados e testados em plataformas específicas (ou seja, linguagem de

Figura 27 – Visão modular da abordagem



Fonte: Autoria própria

programação Java e *Spring Framework*) pela facilidade de desenvolvimento e preservação da *stack* de tecnologia do grupo de pesquisa do orientador deste trabalho. No entanto, vale destacar que os módulos responsáveis pelo *design* podem ser instanciados em outras plataformas e linguagens de programação distintas. A seguir são apresentadas as características e objetivos de cada um dos módulos.

`Features`. Neste módulo estão disponíveis as entidades responsáveis por representar e extrair atributos dos conjuntos de dados brutos. Além disso, esse módulo possui também entidades que auxiliam na exportação dos atributos extraídos em conjuntos de dados brutos, os quais são utilizados mais tarde no processo de mineração de dados.

`Classifier`. Estão disponíveis neste módulo, entidades responsáveis por classificar instâncias a partir de um conjunto de dados preparados. Para isso, esse módulo possui como dependência os módulos `Features` e `weka`. O primeiro é necessário para instanciar entidades do conjunto de dados preparados e o segundo possui entidades que representam os algoritmos de classificação disponíveis no *workbench Weka* (do inglês, *Weikato environment for knowledge analysis*).

`MAPE`. As entidades disponíveis neste módulo dão suporte a instanciação dos *loops* de controle MAPE, onde por meio de classes abstratas e interfaces permitem que novos *loops* sejam instanciados para lidar com futuras ameaças/vulnerabilidades em escala.

`LibSelfProtection`. Diferente dos outros módulos que fazem parte do módulo principal `SelfProtection`, este módulo é executado no espaço de memória de uma SApp. Neste sentido, devem estar disponíveis neste módulo entidades responsáveis por instanciar os pontos de monitoramento e efetores da SApp. Por este motivo, esse módulo pode apresentar baixa capacidade de generalização em outras plataformas e linguagens de programação que sejam diferentes das utilizadas no desenvolvimento de uma SApp.

`SApp`. Este módulo possui entidades, bibliotecas, imagens e toda infraestrutura que fazem parte de uma SApp.

`AppControlPanel`. Por meio de interfaces gráficas, este módulo disponibiliza funcionalidades automatizadas dos módulos `Features`, `Classifier` e `MAPE`. Em resumo, essas funcionalidades apoiam os processos de geração de dados brutos, exportação do conjunto de dados preparados e monitoramento dos serviços de autoproteção estejam disponíveis e funcionais aos especialistas que participam das atividades previstas na abordagem proposta.

`AppMape`. Este módulo contém os *loops* MAPE instanciados que são disponibilizados como serviços ao SApp por meio de APIs REST. De modo similar ao módulo `AppControlPanel`, os atores envolvidos no projeto de uma SApp podem operar este módulo por meio de interfaces gráficas.

`DataBase`. Estão disponíveis neste módulo as entidades especializadas no acesso aos dados armazenados referentes a uma SApp, sendo responsável por persistir instâncias de dados brutos e tratados. Esses dados são provenientes das **Atividades 1** e **2** mencionadas nesta seção, além de configurações referentes ao módulo `AppControlPanel`. Cabe destacar que esse módulo utiliza a biblioteca `Spring Data`[1] para acesso ao banco de dados, permitindo que outros sistemas gerenciadores de banco de dados sejam utilizados em substituição ao MongoDB[2], que foi utilizado neste trabalho (Veja Capítulo 5).

`weka`. Este módulo contém uma API para linguagem Java que permite utilizar funcionalidades de mineração de dados disponíveis no *workbench* Weka[3]. Para isso, são disponibilizados componentes e entidades responsáveis por representar instâncias de conjuntos de dados, pré-processamento de dados, classificação e clusterização de dados processados, avaliações de modelos de classificação e clusterização, seleção de atributos, entre outros.

`SpringData` Este módulo disponibiliza um modelo de programação baseado na linguagem Java com base no *framework* Spring[4] com alto nível de abstração para acesso e manipulação a variados tipos de repositórios de dados, sendo estes relacionais ou não relacionais.

[1] https://spring.io/projects/spring-data
[2] https://www.mongodb.com
[3] https://www.cs.waikato.ac.nz/ml/weka/
[4] https://spring.io/

A seguir, as Seções 4.1.1 a 4.1.4, são detalhadas cada umas das atividades que fazem parte da abordagem proposta neste trabalho de mestrado, além dos seus processos, atividades e componentes de *design* e *runtime*.

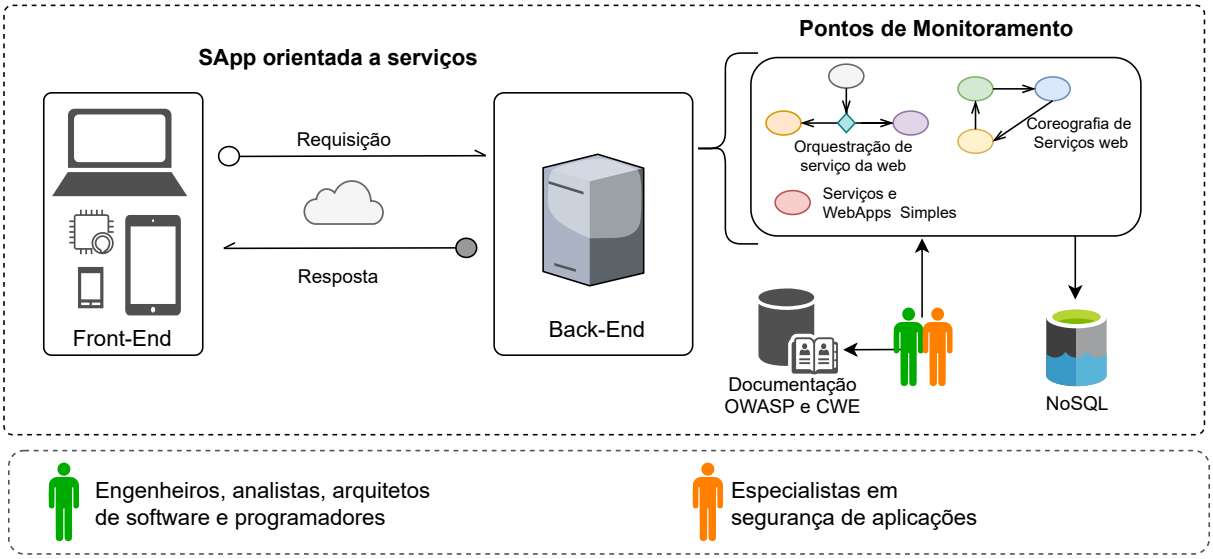
4.1.1 Atividade 1

De acordo com Sommerville (2019), a segurança em sistemas se baseia em três noções fundamentais. A primeira está relacionada à prevenção de vulnerabilidades, onde estratégias são utilizadas para que problemas de segurança sejam evitados. A segunda está relacionada a detectar e repelir ataques, sendo a técnica de monitoramento essencial para verificar a ocorrência de padrões incomuns no sistema. E por fim, a terceira se destina a recuperação do sistema quando ocorrerem problemas. Neste contexto, para atender a primeira noção de segurança, é necessário que processos específicos sejam adotados durante o ciclo de vida de desenvolvimento, como avaliações de riscos e fraquezas e diretrizes de segurança, que devem acompanhar o projeto de arquitetura e codificação da aplicação. A segunda e terceira noção de segurança supracitadas podem ser alcançadas por meio de propostas de *design*, onde pontos de monitoramento são definidos em componentes e camadas estratégicas da aplicação, para que processos automatizados presentes nessa abordagem sejam executados.

A Figura 28 ilustra a **Atividade 1**, onde ocorre o desenvolvimento de uma SApp em tempo de *design*, que busca atender a noção de prevenção de vulnerabilidades e definir pontos de monitoramento, para permitir detectar e repelir futuros ataques em SApps. Portanto, engenheiros de software devem considerar ações maliciosas intencionais, que podem ser internas ou externas, além daquelas que são provenientes de erros originados do processo de desenvolvimento. De acordo com Sommerville (2019), aplicar segurança em todos os componentes e camadas de uma aplicação é dispendioso e pode atrasar a entrega do projeto. Logo, a avaliação de risco é uma técnica que pode viabilizar a aplicação de segurança nos pontos essenciais de um projeto de software, onde riscos e fraquezas são modelados com base em diretrizes de segurança (OWASP, 2021e). Neste contexto, engenheiros de software podem definir quais componentes e/ou camadas da aplicação merecem atenção para que os pontos de sensoriamento sejam implementados, os quais fornecerão informações para o módulo monitor e efetores da aplicação protetora em tempo de execução (fase de *runtime*).

A avaliação e gerenciamento de riscos de uma aplicação, envolve processos que devem acompanhar todo o seu ciclo de vida, estando presente desde a especificação inicial até seu uso operacional. As etapas da avaliação de riscos são: (i) avaliação preliminar de risco, onde o objetivo é identificar riscos genéricos aplicáveis ao sistema e decidir se um nível de segurança adequado pode ser alcançado através de um custo razoável; (ii) avaliação de risco do projeto, que ocorre durante o ciclo de vida de desenvolvimento, onde é levado em conta detalhes técnicos

Figura 28 – Atividade 1 - Visão Geral



Fonte: Autoria própria

do projeto, além de implementação e codificação; e (iii) avaliação de risco operacional, que concentra-se no uso do sistema, onde riscos que venham a ser identificados, retroalimentam novos requisitos de segurança em futuras implementações (SOMMERVILLE, 2019).

De acordo com as descobertas reportadas no Capítulo 3, no geral, arquitetos de software, analistas e desenvolvedores de sistemas não apresentam cultura estabelecida em relação ao tratamento de cibersegurança no projeto dos sistemas de software. Portanto, visando contornar tal carência, recomenda-se que especialistas em segurança voltados para a área de aplicações estejam presentes na fase de projeto de uma SApp. Nessa direção, Sommerville (2019) apresenta um conjunto de diretrizes que podem dar apoio a fase de projeto, onde busca-se um nível de organização que permita que ativos críticos possam ser protegidos contra ataques externos, e, em paralelo, que esses ativos do sistema sejam distribuídos de modo a minimizar ataques bem sucedidos. As descobertas do capítulo supracitado também revelaram que algumas diretrizes podem nortear a fase de codificação, tais como: utilizar *frameworks* de segurança, preocupar-se com as permissões de acesso a base de dados, validar adequadamente todas as entradas de dados, implementar controle de acesso e registros de *log*, entre outros. Nessa direção, a abordagem proposta neste trabalho sugere que sejam utilizadas as diretrizes reportadas em Sommerville (2019) na fase de projeto de arquitetura e o modelo apresentado por OWASP (2021d) na fase de codificação. A seguir, dez diretrizes relacionadas ao projeto de arquitetura de sistemas são apresentadas (SOMMERVILLE, 2019):

1. **Basear as decisões de segurança da informação em políticas explícitas**. Políticas de segurança definem em alto nível "o que" deve ser defendido. Portanto, decisões de *design*

e requisitos de segurança devem ser levantados com base nessas políticas.

2. **Utilizar defesa em profundidade**. A fim de evitar uma falha geral de sistema, vários mecanismos e técnicas de segurança são adotados como meio de redundância.

3. **Falhar com segurança**. Em casos de falhas da aplicação, mecanismos de segurança devem ser capazes de permitir que o sistema não se torne vulnerável.

4. **Balancear segurança e usabilidade**. Os mecanismos de segurança adotados em um sistema não devem comprometer o uso do sistema por parte dos usuários.

5. **Registrar ações do usuário**. Sempre que possível o sistema deve manter um histórico de ações e comandos efetuados pelos usuários no sistema. Deste modo, futuras auditorias de segurança podem ser praticadas.

6. **Utilizar redundância e diversidade para diminuir riscos**. Na medida do possível, deve-se manter mais de uma versão do software ou base de dados em execução, sendo em diferentes versões e plataformas.

7. **Especificar o formato das entradas do sistema**. Os dados de entrada devem ser especificados para os tipos esperados, evitando, por exemplo, injeção ou modificação dos dados.

8. **Compartimentalizar os ativos**. Os usuários devem obter acesso apenas aos dados ou componentes que realmente sejam necessários, evitando assim que informações sejam perdidas ou corrompidas desnecessariamente em uma situação de ataque.

9. **Projetar para implementação**. O software deve fornecer recursos aos usuários e administradores de sistema para que sejam reduzidas as chances de falhas na fase de implantação.

10. **Projetar para recuperação**. Devem ser projetados mecanismos que permitam ao sistema restaurar-se a um estado operacional e seguro em caso de falhas ou ataques de sucesso.

Na sequência são apresentadas as dez diretrizes de segurança para codificação sugeridas em OWASP (2021d):

1. **Definir requisitos de segurança**. Devem ser definidos requisitos de segurança, que são necessários para garantir que uma das muitas propriedades de segurança diferentes do software sejam satisfeitas. Geralmente os requisitos são derivados de documentos de referências, leis e histórico vulnerabilidades passadas. Nessa direção, a abordagem proposta neste trabalho recomenda o uso do documento de referência "*OWASP Application Security Verification Standard*" (OWASP, 2021b)

2. **Aproveitar *frameworks* e bibliotecas de segurança**. Desenvolvedores de software devem aproveitar *frameworks* e bibliotecas de segurança a fim de cumprir metas e requisitos de segurança. Neste contexto, devem ser aplicadas as seguintes práticas: utilizar bibliotecas e *frameworks* ativos e que sejam amplamente utilizados pelo mercado; manter atualizado um inventário com todas as bibliotecas de terceiros; manter os componentes de terceiros sempre atualizados; e encapsular os componentes de terceiros, mantendo exposto apenas os comportamentos necessários.

3. **Prover acesso seguro ao banco de dados**. Devem ser mantidos seguros os acessos aos repositórios de dados, levando em conta consultas, configurações, autenticação e comunicação.

4. **Implementar fuga de dados**. Esta técnica envolve adicionar caracteres especiais no texto de entrada para evitar que injeções de código sejam interpretadas pelo sistema.

5. **Validar todas as entradas**. Garante que apenas dados formatados corretamente possam entrar com componentes internos do software, onde são avaliadas a sintaxe e semântica dos dados de entrada.

6. **Implementar identidade digital**. Por meio de uma representação única de usuários, transações *on-line* devem ser gerenciadas. Em outras palavras, o servidor deve manter o estado de autenticação dos usuários.

7. **Impor controles de acesso**. Um sistema deve permitir ou negar acesso aos recursos por ele fornecidos, além de conceder ou revogar privilégios aos seus usuários.

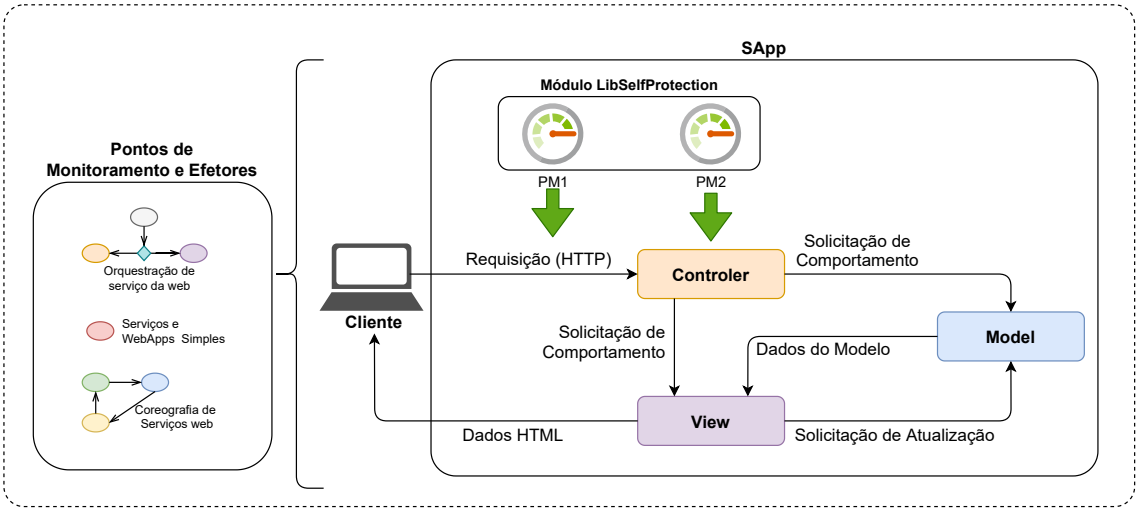
8. **Proteger dados em todos os lugares**. Envolve técnicas que criptografam os dados em trânsito, ou seja, durante uma conexão ou em repouso, quando estão armazenados em alguma unidade.

9. **Implementar registro e monitoramento de segurança**. Devem ser registradas informações de segurança durante um período em que a aplicação estiver em execução.

10. **Manusear todas as falhas e exceções**. Devem ser implementados tratamentos de exceções em todas as áreas do sistema, como regras de negócio, recursos de segurança e código estrutural.

Com base nas fraquezas detectadas durante a avaliação de riscos, pontos de monitoramento e efetores são definidos e implementados na aplicação (SApp). Como ilustrado na Figura 29, a abordagem proposta neste trabalho prevê a criação de dois pontos de monitoramento, sendo um responsável por monitorar o perímetro da aplicação (PM1) e o outro os componentes internos

Figura 29 – Atividade 1 - Pontos de Monitoramento



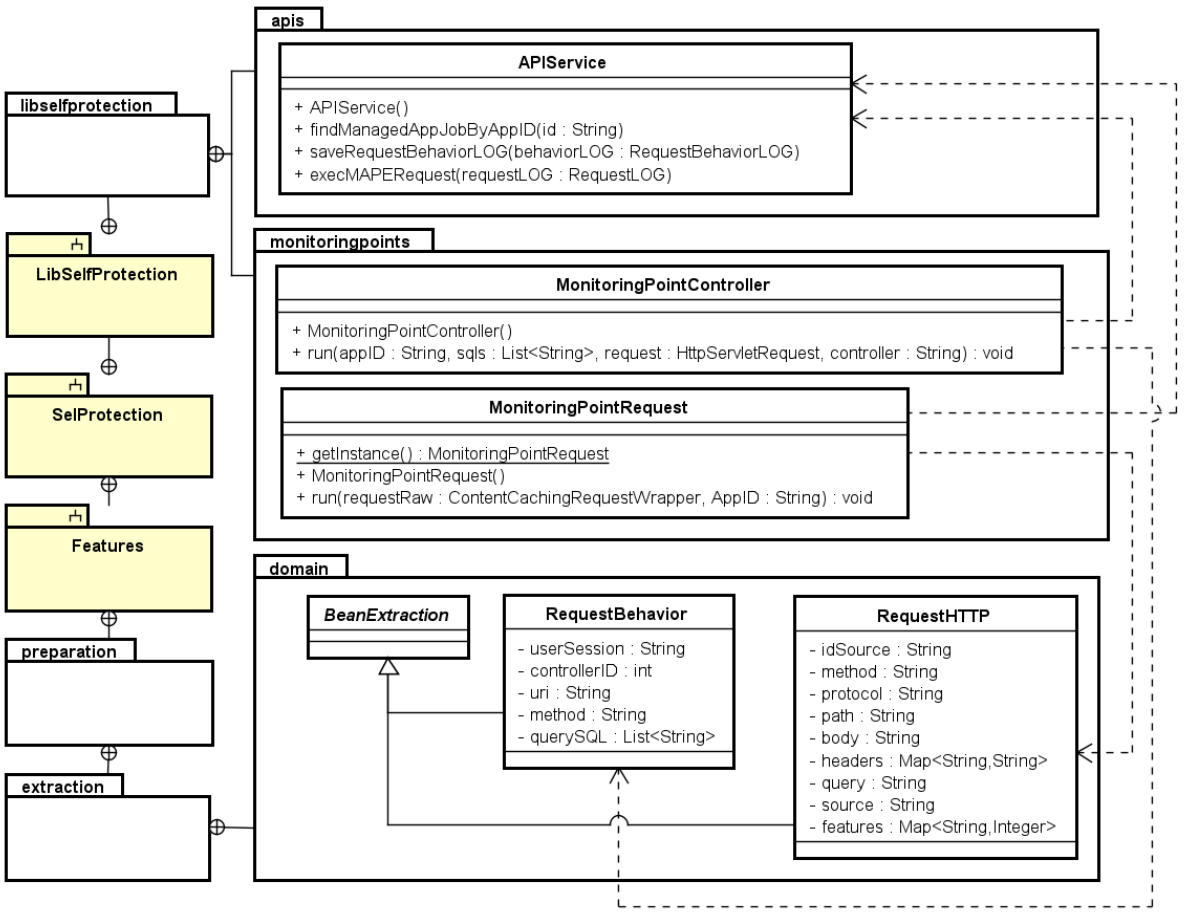
Fonte: Autoria própria

(PM2) da mesma. Em síntese, o PM1 coleta os dados contidos nas requisições HTTP/HTTPS [5], a saber: métodos de requisição (POST, GET, PUT, DELETE, entre outros), protocolo (HTTP ou HTTPS), *path* do recurso requisitado, *paylod*, dados do cabeçalho e parâmetros. Já o PM2 explora o comportamento do componente `Controller` presente na arquitetura MVC (do inglês, *Model-View-Controller*) da aplicação. Neste contexto, retomando à QP3 (veja Seção 3.2), estes pontos de monitoramento se destacam da maioria dos trabalhos extraídos no mapeamento apresentado no Capítulo 3, que tratam de vulnerabilidades nas camadas de rede e física. Portanto, esta abordagem permite que falhas de arquitetura e codificação possam ser detectadas e tratadas em tempo de execução, sendo que os PMs avaliam comportamentos dos componentes internos da aplicação. Além disso, é importante ressaltar que os pontos de monitoramento PM1 e PM2 podem ser reescritos para outras plataformas e linguagens de programação, sem que alterações sejam realizadas nos *loops* MAPE. Essa condição é alcançada devido à comunicação dos PMs com o módulo de autoproteção ocorrer por meio de APIs REST, sendo um protocolo leve e multiplataforma. Assim, em ambos os pontos de monitoramento, os dados coletados são armazenados em uma base de dados NoSQL para que as atividades seguintes usufruam dos dados para mineração e/ou criação/atualização dos modelos de classificação.

A Figura 30 ilustra o diagrama de classes para lidar com os pontos de monitoramento PM1 e PM2. O pacote `domain` contém classes responsáveis por representar os dados coletados pelos pontos de monitoramento. A classe `RequestHTTP` é responsável por representar os dados do PM1 e a classe `RequestBehavior` representa os dados do PM2 (veja Figura 29). Em `RequestHTTP`, os atributos *method, protocol, path, body, headers e query* são dados existentes nas requisições HTTP/HTTPS. O atributos *idSource* e *features* representam consecutivamente o identificador

[5] HTTPS (do inglês, *Hyper Text Transfer Protocol Secure*)

Figura 30 – Atividade 1 - *Beans* dos pontos de monitoramento



Fonte: Autoria própria

da requisição e os futuros atributos que serão extraídos na **Atividade 2**, quando é iniciado o processo de aprendizagem. Na classe `RequestBehavior`, seus atributos são extraídos dos controladores requisitados em uma SApp, onde o atributo *userSession* representa a seção do cliente que efetuou a requisição, *controllerID* representa o identificador do controlador, *uri* o endereço do recurso disponível no controlador, *method* o método utilizado para fazer a requisição (por exemplo, GET e POST) e *querySQL* as instruções SQL disparadas pelo controlador ao banco de dados responsável por armazenar os dados do SApp. As classes contidas no pacote `monitoringpoints` são responsáveis por fazer a comunicação entre uma SApp e o módulo `AppMape` (veja Figura 27), onde a entidade `MonitoringPointController` é responsável pelo PM1 e `MonitoringPointRequest` pelo PM2. O pacote `apis` contém a entidade utilitária `APIService`, que é responsável por estabelecer as requisições REST entre a SApp e o módulo `AppMape`.

Em relação aos efetores, pontos de variabilidade devem ser projetados, modelados, implementados e catalogados para permitir que o software protegido (SApp) seja adaptado em

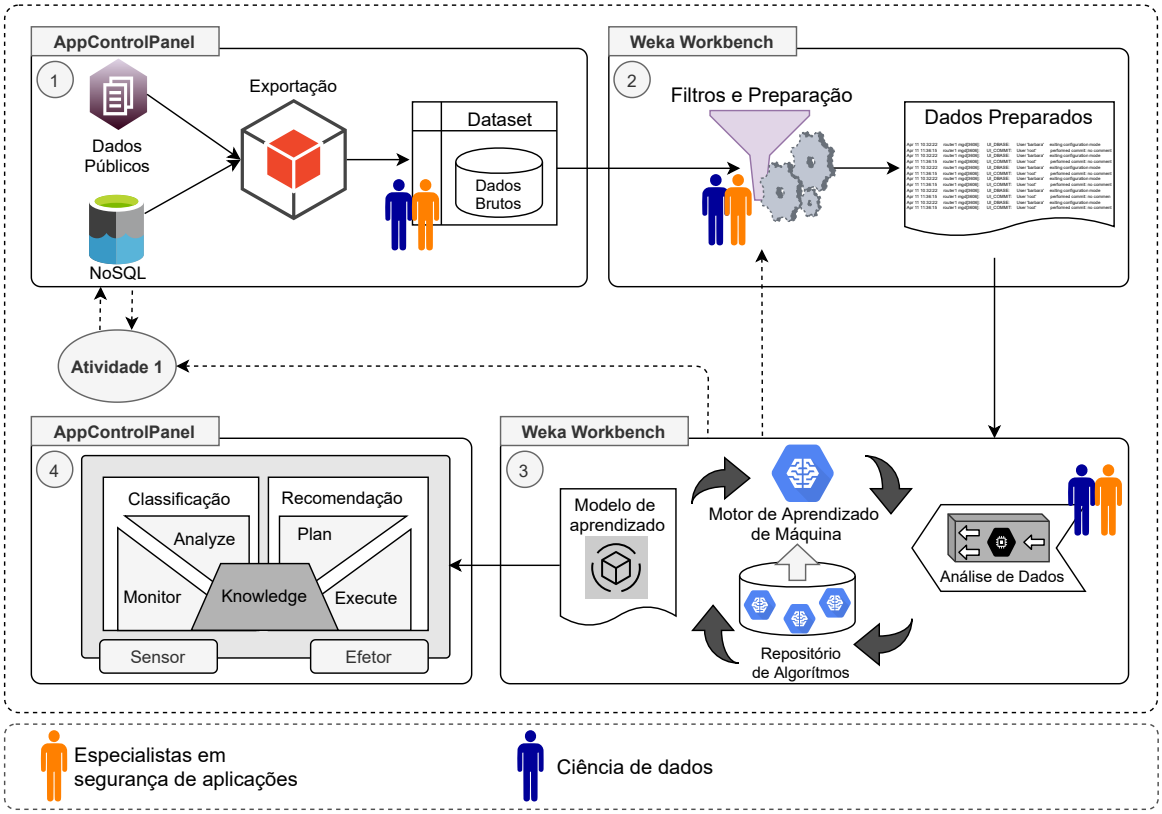
tempo de execução. Nesse sentido, unidades algorítmicas e estruturais, conhecidas na literatura como *features*, são substituídas sem que o sistema seja interrompido e reiniciado, como meio de contra medida aos ataques e ameaças detectados (FERBER; HAAG; SAVOLAINEN, 2002). Embora este trabalho também trate as características extraídas de *datasets* no processo de mineração de dados como *features*, o termo possui significado e aplicações diferentes neste contexto. Para viabilizar o uso dos efetores, a abordagem tratada por este trabalho sugere que as *features* (pontos de variabilidade) sejam mapeados com itens arquiteturais do SApp e sejam registrados na base de dados da aplicação protetora (repositório NoSQL). Em paralelo deve ser apontado a ordem de prioridade da implantação de tais *features*, de modo a garantir que conflitos sejam evitados no momento da adaptação (ELEUTéRIO, 2017).

4.1.2 Atividade 2

Nesta fase ocorre o processo de aprendizagem e implantação dos modelos de classificação nos *loops* MAPE, atividade que ocorre durante o *design* de uma SApp. Em resumo, conforme ilustrado na Figura 31, essa atividade é composta por quatro tarefas, a saber: (1) extração de *dataset*; (2) aplicação de filtros e preparação de dados; (3) análise de dados e criação do modelo de classificação; e (4) instanciação do modelo de classificação em um *loop* MAPE-K. De acordo com nossas descobertas (veja Capítulo 3), essas fases devem ser conduzidas por profissionais com conhecimento em técnicas de ciência de dados e cibersegurança voltada para a camada de aplicação (do inglês, *Application Security* – AppSec).

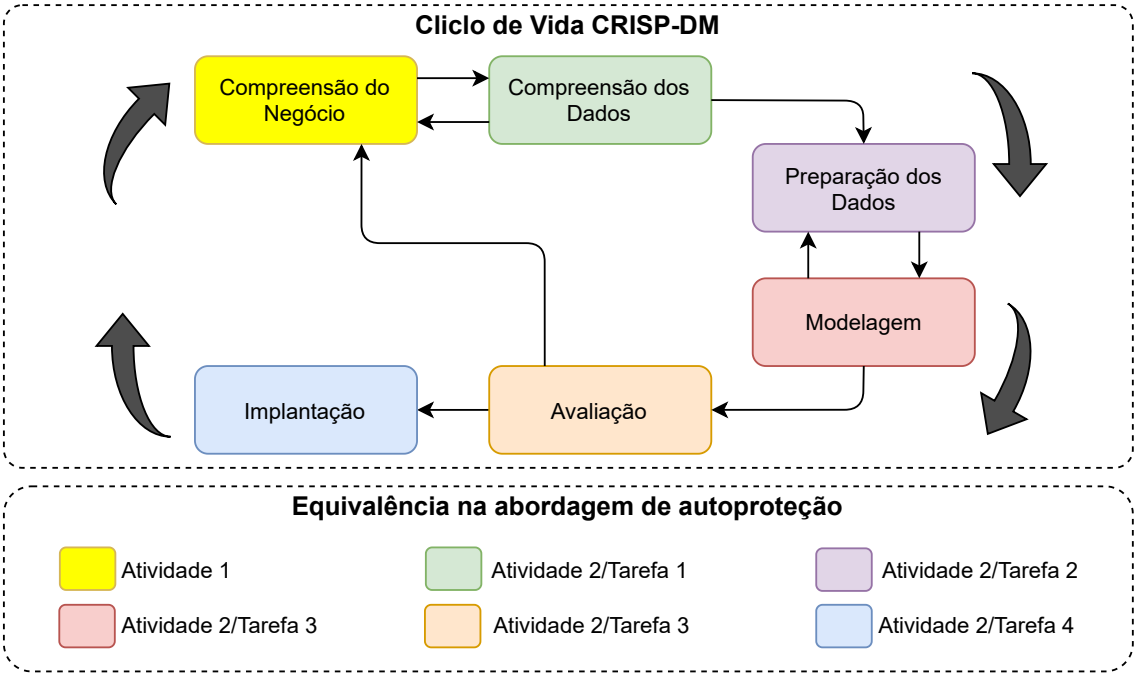
O fluxo das **Tarefas 1, 2, 3 e 4** da abordagem proposta neste trabalho foram baseadas no ciclo de vida CRISP-DM, que é ilustrado na Figura 32. De acordo com Witten *et al.* (2011), inicialmente ocorre a ***Compreensão do Negócio***, ou seja, busca-se investigar os objetivos e requisitos necessários. Essa fase é equivalente a **Atividade 1** da abordagem proposta (veja Figura 26), onde são estudados e avaliados os possíveis pontos de monitoramento de uma SApp. A próxima fase do ciclo de vida é ***Compreensão dos Dados***, onde um conjunto de dados inicial é estabelecido e estudado para garantir que as próximas fases serão viáveis. Se estes dados forem classificados como inadequados, ajustes podem ser necessários na fase anterior. Este ponto do ciclo é compatível com a **Tarefa 1**, onde dados são gerados e avaliados por especialistas em ciência de dados e (ciber)segurança. A próxima fase (***Preparação dos Dados***) é compatível com a **Tarefa 2**, onde ocorre o processamento dos dados brutos para que os algoritmos de aprendizado de máquina possam produzir um modelo. Na sequência, a fase ***Modelagem*** é equivalente a **Tarefa 3**, em que o modelo de classificação é construído. Neste ponto do ciclo, os resultados obtidos durante a modelagem podem fornecer novos **insights**, o que justifica diversas iterações com a **Tarefa 2**. Na fase de ***Avaliação*** que está incluída ainda na **Tarefa 3**, podem ser aplicadas diversas métricas para avaliar o modelo de classificação gerado. Novos *insights* podem apontar a necessidade de reavaliar a compreensão do negócio, o que na abordagem proposta neste trabalho

Figura 31 – Atividade 2 - Visão Geral



Fonte: Autoria própria

Figura 32 – Atividade 2 - Ciclo de vida CRISP-DM



Fonte: Adaptado de (WITTEN *et al.*, 2011)

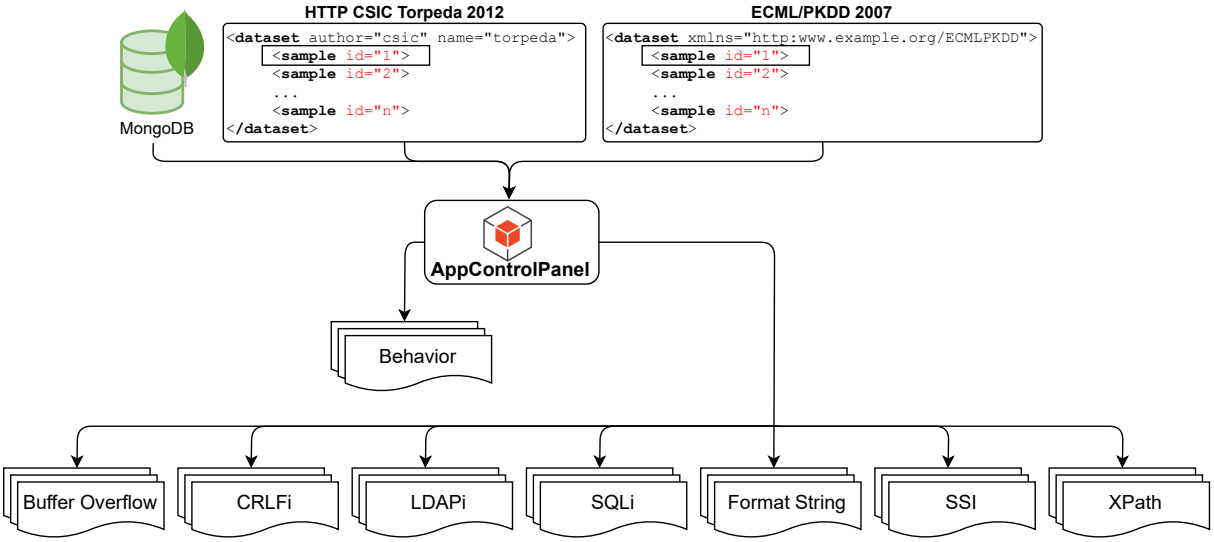
significa rever os pontos de monitoramento na **Atividade 1**. Por fim, a fase de ***Implantação*** é semelhante a **Tarefa 4**, onde ocorre a implantação do modelo de classificação no ambiente de execução. A seguir são detalhadas cada uma das tarefas.

A **Tarefa 1** deve iniciar quando versões funcionais de uma SApp estiverem disponíveis. Quando acessadas, por meio de requisições HTTP/HTTPS válidas e maliciosas, essa aplicação produz *logs* de acesso que podem ser armazenados pela infraestrutura disponibilizada pela abordagem proposta neste trabalho (veja Seção 4.1). Posteriormente, esses *logs* irão compor o conjunto de dados de treinamento para cada tipo de ataque. Sobre essa tarefa, vale destacar que as requisições válidas e maliciosas podem ser executadas manualmente ou a partir de processos automatizados, tais como testes de penetração para os acessos maliciosos e validações de dados para acessos (in)válidos. Além disso, cabe destacar também que *datasets* públicos podem ser utilizados para alguns tipos de ameaças que são de conhecimento comum na maioria das aplicações (SApp). Conforme reportado no Capítulo 3 e nas diretrizes apresentadas na Seção 4.1.1, para detectar alguns comportamentos maliciosos específicos é necessário que o *dataset* de treinamento seja gerado a partir da própria aplicação (SApp). De acordo com as evidências reportadas no capítulo supracitado, isso ocorre devido a necessidade de se observar características específicas de uma SApp que não seria possível apenas com um *dataset* genérico para um tipo de ataque.

A Figura 33 ilustra a atividade de exportação de *datasets* que devem ser utilizados para a criação de modelos de classificação para diferentes tipos de ameaças/vulnerabilidades. Em síntese, esse processo é composto de duas fases, sendo primeira realizada pela obtenção de *datasets* que representam comportamentos da aplicação (SApp) resultantes dos pontos de monitoramento (PM1 e PM2). Já a segunda é realizada pelo módulo `AppControlPanel` (veja Figura 27), que permite extrair atributos e características a partir dos *datasets* de comportamentos e gerar novos *datasets*, que são chamados de *datasets* de *features*. Vale ressaltar que cada *dataset* de *features* está relacionado diretamente a um tipo de vulnerabilidade/ameaça. Os *datasets* de comportamentos são formados a partir dos dados públicos "HTTP CSIC Torpeda 2012"[6] e "ECML/PKDD 2007"[7], além dos dados gerados a partir das requisições realizadas a uma SApp, que são armazenadas na base de dados da aplicação protetora (componente MongoDB). Diante do exposto, vale destacar que os *datasets* de *features* são capazes de gerar modelos de classificação para detectar ameaças/vulnerabilidades presentes nos relatórios da OWASP (veja Figura 4), a saber: *Buffer Overflow*, *CRLFi*, *Format String*, *LDAPi (do inglês, LDAP injection)*, *SQLi (SQL Injection)*, *SSI*, *XPath*, *XSS (Cross-site scripting)*. Já os *datasets* gerados a partir do PM2 (componente `Behavior`) são capazes de detectar comportamentos anômalos nos componentes `Controller` de uma SApp. Especificamente sobre esse último *dateset*, é importante salientar

[6] https://www.tic.itefi.csic.es/torpeda/
[7] http://www.lirmm.fr/pkdd2007-challenge

Figura 33 – Atividade 2 – Exportação de *datasets*



Fonte: Autoria própria

que futuras medidas contra ameaças/vulnerabilidades pode exigir a implementação de novas extrações de atributos e características. Contudo, o *design* aplicado na abordagem proposta neste trabalho é capaz de suportar tais implementações, facilitando a escalabilidade da mesma.

No que diz respeito ao processo de extração de *features* (ou seja, atributos e características) dos *datasets* de comportamentos, cabe destacar que o mesmo ocorre de maneira distinta para os dados gerados a partir de comportamentos do perímetro da aplicação e de componentes internos de uma aplicação (SApp). Para os comportamentos de perímetro é avaliado o conteúdo de cada requisição HTTP/HTTPS, onde são contabilizados a ocorrência de palavras chaves e caracteres que são comuns para cada tipo de ataque. Para isso, existe um conjunto de caracteres e palavras chaves para cada tipo de ataque que pode fornecer evidências sobre as ameaças/vulnerabilidades que uma SApp possa estar sofrendo (BHAGWANI *et al.*, 2019; MARTINS *et al.*, 2021). Em paralelo, a extração de *features* de componentes internos ocorre por meio de um conjunto requisições realizadas para uma SApp, as quais são direcionadas a um componente `Controller` (veja a Figura 29) e uma sessão de usuário específica (ou seja, individual) dentro de um intervalo de tempo considerado como ideal. Nesse caso, o tempo é um elemento relevante para o cenário de ataques, pois as ferramentas que fazem ataques a uma aplicação (SApp) normalmente executam um conjunto de requisições (ou seja, "rajadas") fora do comportamento padrão de uma aplicação. Para descrever o comportamento do componente `Controller`, o ataque injeção de SQL (SQLi) será considerado. Em geral, esse componente contabiliza a quantidade de requisições para cada método do protocolo HTTP/HTTPS (`GET`, `HEAD`, `POST`, `PUT`, `DELETE`, `CONNECT`, `OPTIONS`, `TRACE` e `PATCHE`), onde o rol de métodos utilizados pode variar para cada tipo de aplicação. Além disso, pode ser contabilizado a média de *bytes* para cada tipo de instrução SQL, que são

enviadas ao banco de dados, como por exemplo: consultas `SELECT`; comandos para manipulação de dados como `INSERT`, `DELETE` e `UPDATE`; e comandos para definição de dados como `CREATE`, `DROP` e `ALTER`. Em resumo, essas informações, quando combinadas, podem oferecer evidências que permitem identificar, em um primeiro momento, se uma requisição é considerada normal ou anômala. Havendo alguma evidência de anormalidade, essa requisição deve passar, em um segundo momento, por um processo de classificação quanto ao tipo de ataque que pode pertencer.

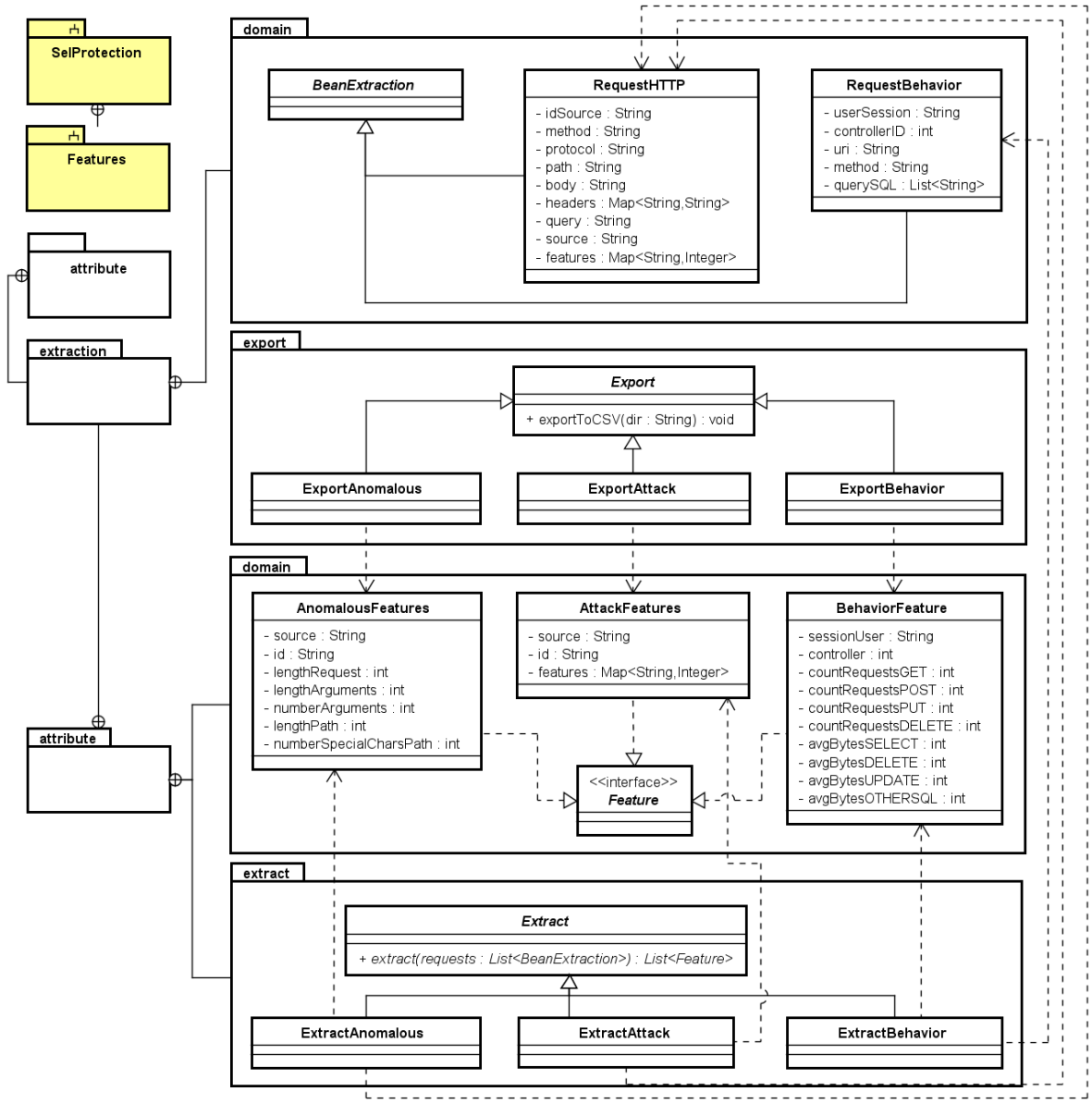
A Figura 34 ilustra o diagrama classes do módulo `Features`, que funciona como utilitário ao módulo `AppControlPanel` para exportar os *datasets* de *features* (veja a Figura 27). O módulo `Features` é composto por quatro pacotes, a saber: (i) `extraction::domain`, que possui entidades *beans* responsáveis por representar os pontos de monitoramento; (ii) `extraction::export`, que contém entidades responsáveis por extrair atributos (também conhecido como *features*) das entidades *beans*; (iii) `feature::domain`, que possuem entidades que representam instâncias dos atributos extraídos das entidades *beans*; e (iv) `feature::extract`, que contém entidades especializadas em exportar os *datasets* de *features* em arquivos no formato CSV (do inglês, *Comma-Separated Values*)[8]. O módulo `AppControlPanel` permite que o especialista em ciência de dados escolha a fonte de dados de comportamentos, sendo conjuntos de dados públicos e/ou dados de *log* do próprio SApp (DBMongo). Como forma de ilustrar a interação entre os objetos, na Figura 35 é ilustrado um diagrama de sequência para extração de dados pelas classes contidas no módulo de `Features`. Inicialmente, Passo 1 ocorre a instância de um objeto (`extract`) da classe `Extract`, que pode ser uma anomalia (classe `ExtractAnomalous`), um ataque (classe `ExtractAttack`) ou um comportamento (classe `ExtractBehavior`). Em seguida, Passo 2, o método de extração de *features* é chamado para que a lista retornada (`List<Feature>`) seja exportada para o alvo desejado (arquivo CSV) nos Passos 3 e 4.

A partir dos *datasets* de *features*, **Tarefas 2 e 3** (veja Figura 31) são conduzidas no *workbench* Weka[9]. Na **Tarefa 2** ocorre a preparação dos dados brutos, onde dados inúteis ou que apresentam algum tipo de ruído que possam prejudicar o comportamento dos modelos são eliminados. Nessa tarefa, atributos podem ser convertidos e filtros também podem ser aplicados para que o conjunto final de dados normatizados seja obtido. Em seguida, os dados gerados passam por um processo de análise (**Tarefa 3**), onde especialistas em ciência de dados avaliam métricas de performance dos dados preparados, como taxa de detecção, falso-positivo, falso-negativo, entre outras métrica. Vale ressaltar que neste ponto, diversas interações entre as **Tarefas 2 e 3** podem ocorrer até que o modelo de classificação seja validado, onde a quantidade de interações dependerá dos critérios de segurança adotados na abordagem de autoproteção. Quando a **Tarefa 3** resultar em um modelo de classificação com performance aceitável, este

[8] Arquivo de texto regulamentado pelo padrão RFC 4180

[9] O *workbench* Weka fornece aos desenvolvedores um ambiente completo para modelagem visual de processos de mineração, além de um conjunto de algoritmos que podem atuar em diferentes cenários desse processo, a saber: preparação, classificação, regressão, clusterização, regras de associação e visualização de dados.

Figura 34 – Atividade 2 - Diagrama de Classes do módulo `Features`

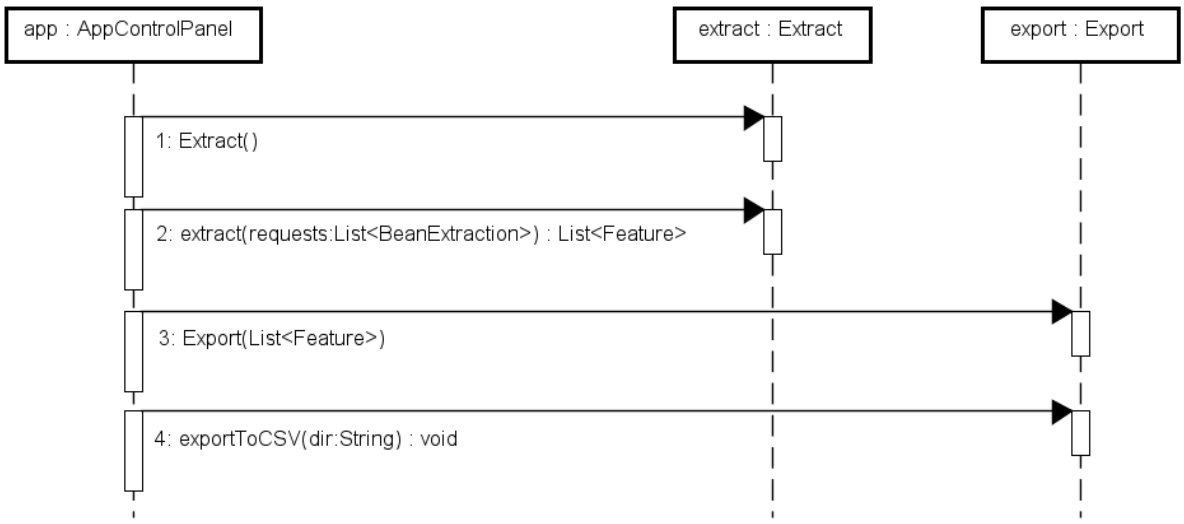


Fonte: Autoria própria

é implantado em um *loop* MAPE-K correspondente (**Tarefa 4**). O Apêndice C apresenta em detalhes os processos do *workbench* Weka utilizados nas Tarefas **2** e **3**.

Fazendo um paralelo entre a **Atividade 2** apresenta nesta seção e os dados extraídos para as questões de pesquisa 2 e 7 (veja Seção 3.2), nota-se que a abordagem proposta neste trabalho se destaca por permitir aplicar mecanismos de proteção em profundidade, onde várias dimensões de segurança podem ser atingidas simultaneamente. Nessa direção, os *datasets* ilustrados na Figura 33, que são extraídos a partir do PM1, PM2 e dados públicos, além de permitir que uma SApp seja protegida, também pode fazer com que preceitos como confiabilidade, autenticidade e disponibilidade sejam atingidos. Deste modo, por meio de um *design* flexível, pode-se dizer que

Figura 35 – Atividade 2 - Diagrama de Sequência



Fonte: Autoria própria

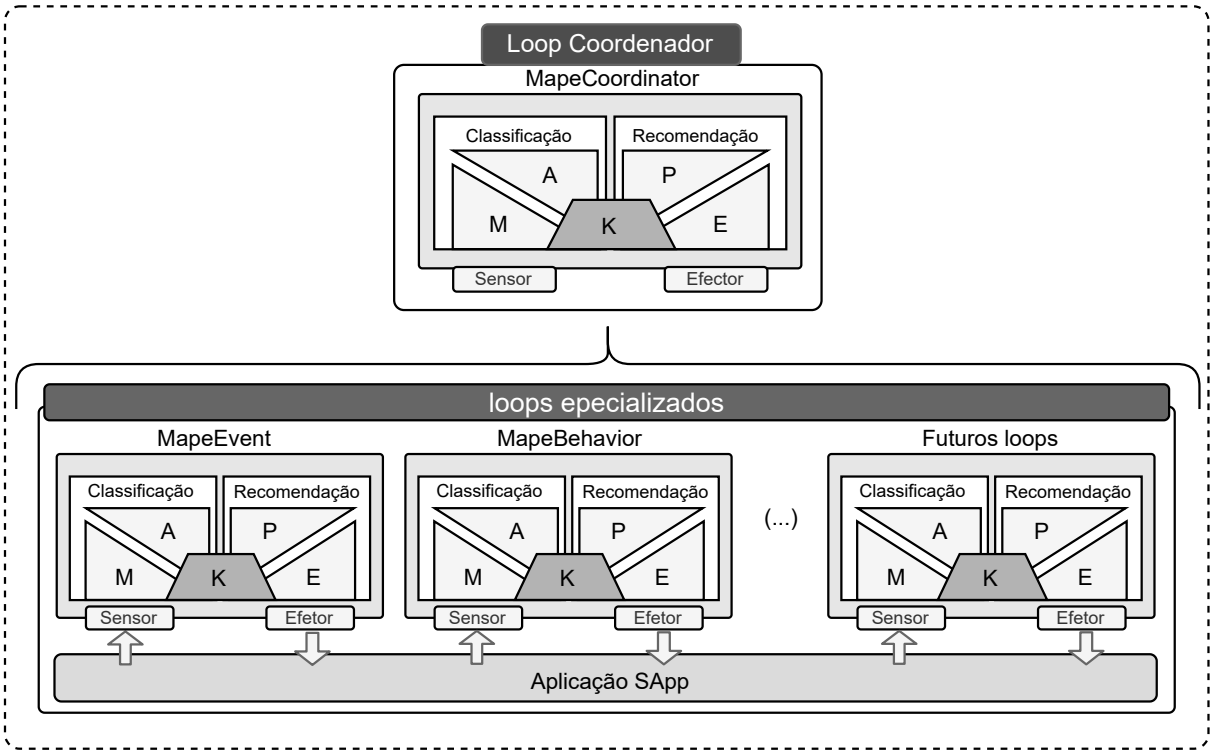
a abordagem proposta permite que novos pontos de monitoramento sejam incorporados para atender as necessidades de domínio de uma nova aplicação.

4.1.3 Atividade 3

A **Atividades 3** ocorre em uma versão estável de uma SApp e em tempo de execução. O módulo protetor atua em uma modalidade de supervisão não intrusiva, onde ocorre o monitoramento dos estados internos e periféricos de uma SApp recupera as informações necessárias para identificação de algum tipo de ameça/vulnerabilidade. Conforme ilustrado na Figura 36, essa atividade foi organizada em um *loop* de controle coordenador que monitora a execução de múltiplos *loops* especializados. Conforme reportado na Seção 4.1.2, cada *loop* especializado é responsável por monitorar um tipo de ameaça, ataque ou comportamento anômalo.

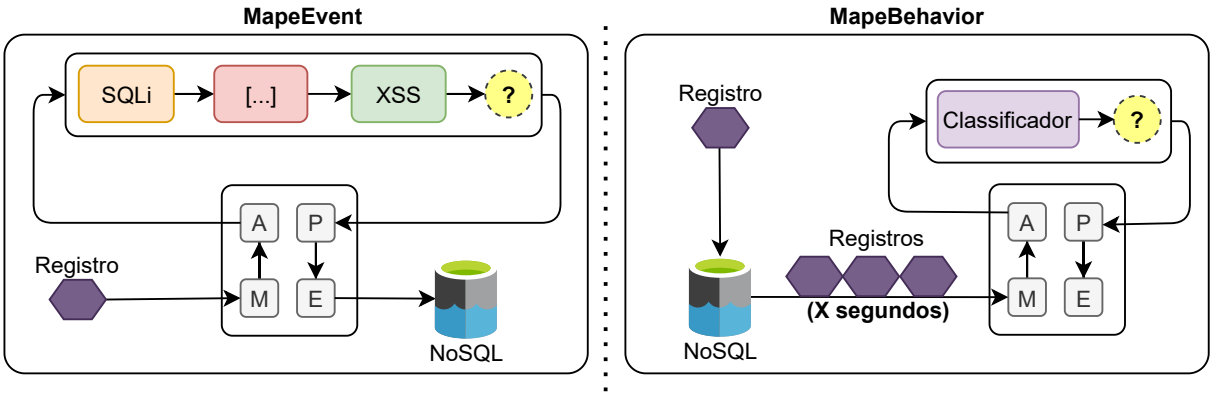
De acordo com a Figura 37, os *loops* `MapeEvent` e `MapeBehavior` possuem características diferentes sobre a maneira como lidam com o fluxo de classificação dos pontos de monitoramento. No *loop* `MapeEvent`, cada requisição HTTP/HTTPS que chega a uma SApp é coletada e classificada proativamente. Neste sentido, podem ser aplicados vários classificadores em sequência na tentativa de identificar alguma anomalia conhecida. Em paralelo, o *loop* `MapeBehavior` pode classificar reativamente comportamentos anômalos em um conjunto de requisições efetuadas a um determinado componente `Controller` (veja a Figura 29). Para isso, o módulo `Monitor` coleta um conjunto de registros em período de tempo estabelecido pelo administrador da aplicação protetora. No que diz respeito ao modo de funcionamento dos *loops* especializados `MapeEvent` e `MapeBehavior`, cabe destacar que eles podem funcionar como um mecanismo de defesa em profundidade para alguns tipos de ataques. O ataque de injeção SQL

Figura 36 – Atividade 3 - Visão Geral



Fonte: Autoria própria

Figura 37 – Atividade 3 - *Loops* MapeEvent e MapeBehavior



Fonte: Autoria própria

(SQLi) pode ser citado como um exemplo para esse tipo de mecanismo de defesa, pois o mesmo é realizado geralmente por meio de ferramentas automatizadas que injetam inicialmente uma série de comandos a fim de compreender os componentes internos de uma aplicação. Em seguida, com base nas informações obtidas, comandos são injetados para efetivar o ataque. Em uma situação como essa, o `MapeBehavior` deve ser capaz de identificar sequências de requisições HTTP/HTTPS ao componente `Controller` que estejam fora do padrão de uma aplicação, enquanto o `MapeEvent` deve avaliar os dados de cada uma das requisições. Em situações como

do exemplo supracitado, o *loop* coordenador tem papel fundamental em organizar os *loops* especializados e as ações dos efetores como, por exemplo, planejar a sequência de possíveis modificações protetivas para a aplicação que está sofrendo ataque.

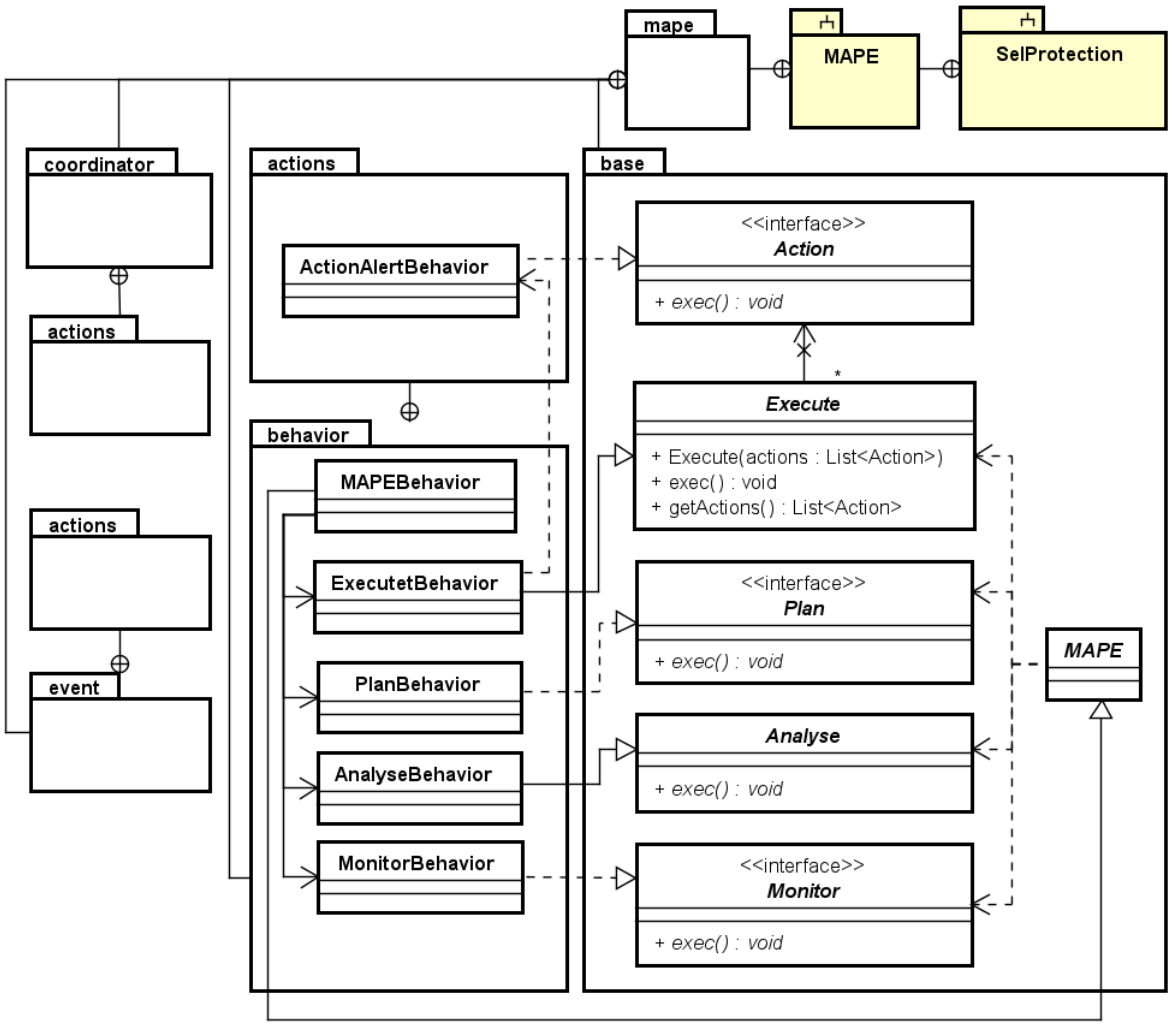
A Figura 38 apresenta um diagrama de classes que ilustra as entidades responsáveis por instanciar novos *loops* MAPE. O pacote `base` contém interfaces e classes abstratas que servem como base para a construção dos *loops* `MapeCoordenator`, `MapeEvent`, `MapeBehavior` e futuros *loops*. As classes `Analyse` e `Execute` e as interfaces `Monitor` e `Plan`, contidas no pacote `base`, são responsáveis por instanciar cada um dos componentes da topologia MAPE, sendo a classe `MAPE` responsável por orquestrar estes componentes. A interface `Action` permite instanciar ações defensivas e protetivas que devem ser executadas pelo componente `Execute`, quando ações maliciosas forem detectadas. Neste contexto, o pacote `behavior` ilustra os componentes do *loop* `MapeBehavior`, que é responsável controlar o PM2 (veja Figura 29). Este *design* se destaca por permitir que a estrutura de classes e interfaces instancie novos *loops* futuramente para diferentes pontos de monitoramento. Os pacotes `coordinator` e `event` são apresentados sem elementos (por exemplo, classes, interfaces) para que a visualização do diagrama não seja prejudicada.

Do ponto de vista operacional, os sensores são responsáveis por capturar parâmetros do ambiente de execução para o sistema supervisor de autoproteção, sendo que na abordagem proposta neste trabalho esse papel é atribuído aos pontos de monitoramento. Os dados coletados pelos sensores são armazenados em um banco de dados NoSQL, que é compartilhado entre os componentes dos loops MAPE, fazendo com que a base de dados se comporte como o componente *Knowledge* (K) da topologia MAPE-K. O componente `Monitor` (M) é responsável por coletar dados da base NoSQL e efetuar a extração de *features* por meio do módulo `Features` (veja a Figura 27). Na sequência, o componente *Analyse (A)* classifica as requisições da aplicação com base no modelo de classificação gerado na fase de *design*. Em seguida, caso alguma ameaça/vulnerabilidade tenha sido identificada, o componente *Plan (P)* elabora um plano de contra medida, onde são definidos, com base no modelo de *features* (pontos de variabilidade), ações a serem aplicadas na aplicação (SApp). Por fim, o componente `Execute` aplica as ações planejadas na aplicação em tempo de execução por meio dos efetores.

A Figura 39[10] ilustra as classes responsáveis pela classificação de ameaças/vulnerabilidades, sendo possível observar a utilização da biblioteca `weka`. A classe abstrata `Classifier` é responsável por instanciar algoritmos de classificação disponíveis no pacote `weka`, que na Figura 39 foram ilustrados pelas classes `J48Adapter`, `NaiveBayesUpdatableAdapter` e `RansomForestAdapter`. Além disso, é possível observar a classe abstrata `Classify`, que permite instanciar classes especializadas na classificação de pontos de monitoramento especí-

[10] Para não comprometer a visualização do diagrama, apenas os principais métodos da classe `Classify` são apresentados.

Figura 38 – Atividade 3 - Diagrama de classes do *loop* MAPE

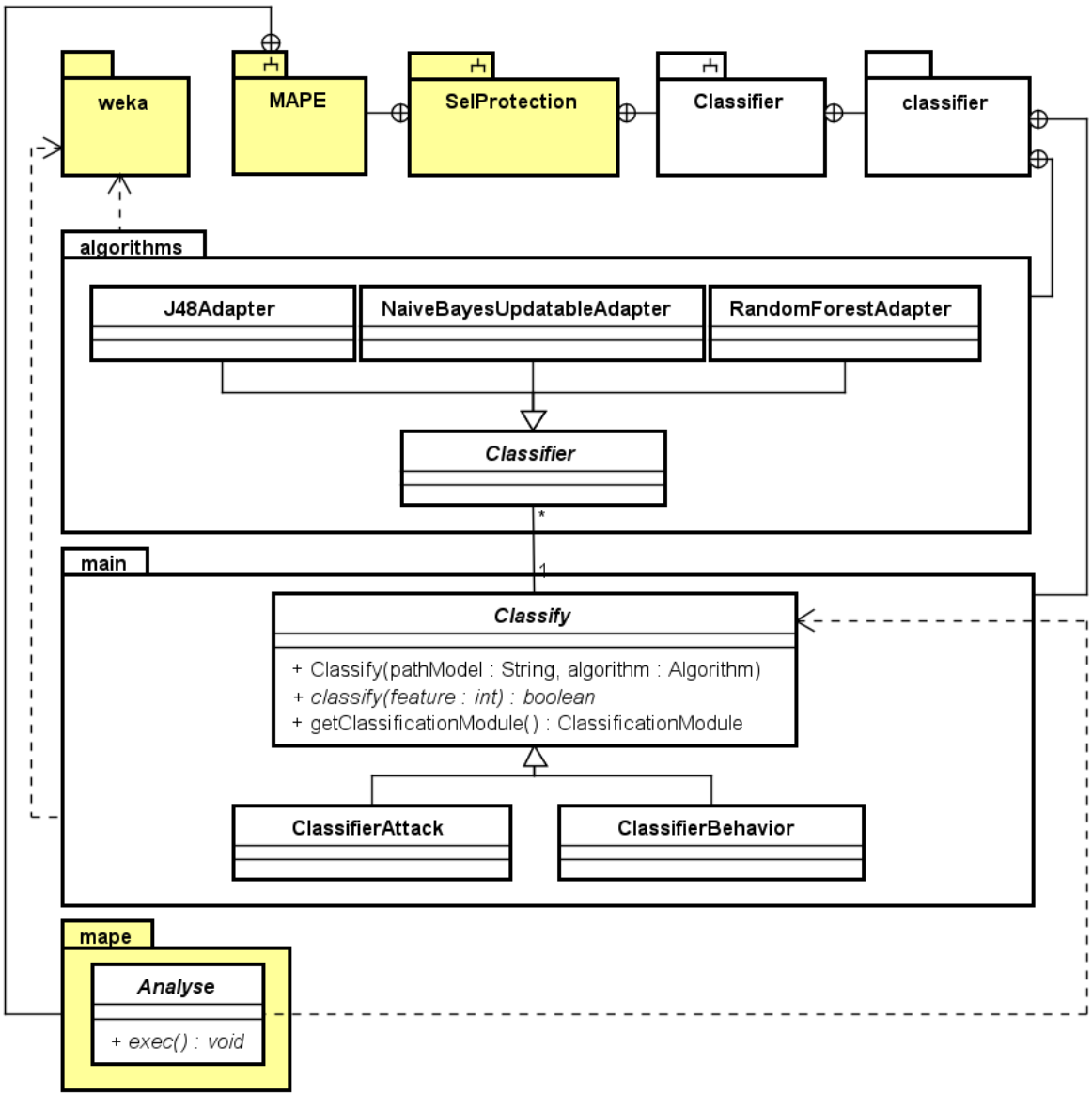


Fonte: Autoria própria

ficos. A classe `ClassifierBehavior` é responsável por classificar dados do PM2 e a classe `ClassifierAttack` é responsável por classificar dados do PM1, conforme ilustrado na Figura 29. Por fim, é importante destacar que novos algoritmos de classificação podem ser incluídos no modelo supracitado apenas pela implementação da interface `Classifier` e classificadores para novos pontos de monitoramento pela implementação da interface `Classify`. Essa estratégia de *design* permite que abordagem proposta possa ser flexível quanto ao número de futuras ameaças/vulnerabilidades que possa lidar no futuro.

Como forma de operacionalizar os módulos `Mape` e `Classifier`, a abordagem proposta neste trabalho oferece o módulo `AppMape`, que por meio de um serviço em *background* é capaz de proteger uma SApp. Nessa direção, o `AppMape` disponibiliza serviços sob o protocolo RESTful, que por meio do módulo `LibSelfProtection` (veja a Figura 27), permite que a SApp se comunique com os *loops* MAPE. Essa arquitetura orientada a serviços permite que

Figura 39 – Atividade 3 - Diagrama de classe do módulo de classificação



Fonte: Autoria própria

a aplicação protetora e protegida (SApp) fiquem isoladas, permitindo que sejam executadas em *hosts* diferentes. Portanto, essa arquitetura de compartimentalização garante que ataques bem-sucedidos a uma SApp não contaminem ou influencie negativamente na aplicação protetora.

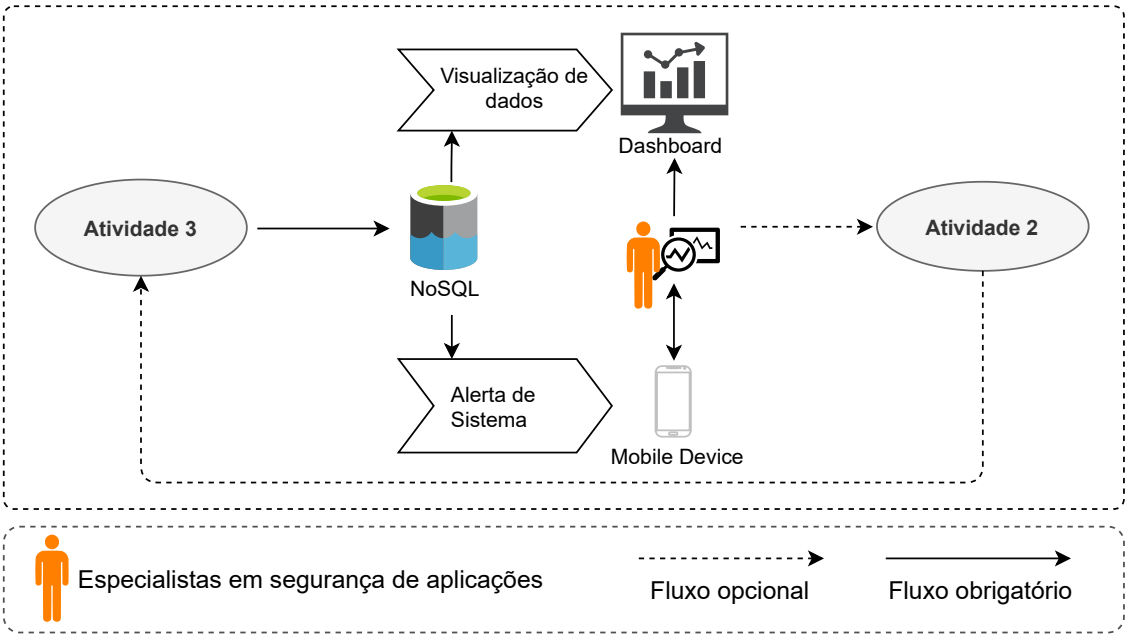
Em comparação com os trabalhos apresentadas no mapeamento da Seção 3.2, a abordagem proposta neste trabalho apresenta iniciativas de *design* para os módulos inteligentes de classificação, além da estrutura de *loop* MAPE para lidar com diferentes tipos de ameaças/-vulnerabilidades. Nesse sentido, conforme pode ser observado na QP5, apenas o trabalho E15 apresenta mecanismo de defesa baseado em *loop* fechado voltado à camada de aplicação. No entanto, o foco do referido trabalho está na defesa da perturbação do modelo inteligente da

aplicação protetora. Em paralelo, conforme evidenciado na QP5.1, 45,05% dos trabalhos não apresentam dados concretos sobre qual arquitetura é empregada em uma solução de autoproteção. Diante desse cenário, pode-se dizer que este trabalho apresenta uma contribuição nessa direção. Ainda neste contexto, de acordo com a QP6, apenas dois trabalhos permitem que o mecanismo de defesa empregue abordagens de defensivas proativas e reativas ao mesmo tempo, algo que também é permitido pela proposta deste trabalho de mestrado.

4.1.4 Atividade 4

Nesta fase ocorre o gerenciamento dos pontos de monitoramento pelo especialista em segurança, que podem ser acompanhados por meio de *dashboards* ou mensagens de alertas via sistema. Em situações que uma SApp se defenda proativamente contra ameaças/vulnerabilidades, o administrador deve acompanhar se os modelos de classificação estão trabalhando de maneira eficiente ou se necessitam ser atualizados/modificados. Nas situações que ocorrem ações de defesa reativa, que geralmente nascem a partir ameaças/vulnerabilidades de dia zero, os modelos de classificação também devem ser atualizados para que futuros ataques da mesma natureza possam ser detectados no futuro. Em ambas as situações, a atualização/modificação dos modelos de classificação significa que a **Atividade 2** deve ser revisitada e os laços MAPE-K devem ser atualizados. A Figura 40 ilustra as atividades referente a **Atividade 4**.

Figura 40 – Atividade 4 - Visão Geral



Fonte: Autoria própria

Na abordagem proposta neste trabalho, os dados que são coletados e classificados pelos

loops MAPE são armazenados na base dados MongoDB da aplicação protetora. Assim, por meio de interfaces gráficas contidas no módulo `AppControlPanel` (veja a Figura 27), é possível visualizar e avaliar cada um dos eventos coletados pelos pontos de monitoramento. De acordo com (SAHATQIJA *et al.*, 2018), as bases de dados NoSQL possuem algumas vantagens em relação às relacionais, as quais são aproveitadas na abordagem proposta. Por exemplo, as bases NoSQL permitem uma escalabilidade horizontal, permitindo atender um aumento de demanda por meio de adição de novos nós para aumentar a capacidade de armazenamento e processamento da aplicação. Esse tipo de abordagem pode tornar os custos com armazenamento de dados mais vantajosas ao com passar do tempo, uma vez que os registros de sensoriamento e classificação dos PMs de uma SApp podem se tornar elevados. Além disso, esse tipo de base permite maior flexibilidade, fazendo com que o esquema de dados seja alterado durante a evolução da aplicação, sem que seja necessário alterar suas estruturas com antecedência. Essa propriedade pode ser necessária para que os esquema de dados armazenados na aplicação protetora possam acompanhar as mudanças ocorrer nos pontos de monitoramento, *features* que são extraídas dos conjuntos de dados de comportamento, e modelos de classificação (**Atividade 2**). Diante do contexto exposto, o uso da base de dados MongoDB[11] torna o módulo `AppControlPanel` mais escalável em relação a performance e flexibilidade de armazenamento, fazendo que os especialistas em segurança de aplicações tenham interfaces de visualização com alta disponibilidade e robustez.

4.2 Recomendações e boas práticas para a aplicação protetora

Além de seguir boas práticas e diretrizes de segurança para o desenvolvimento de uma SApp, recomendações devem ser seguidas nas fases de projeto de arquitetura, desenvolvimento e implantação da aplicação protetora para diminuir os riscos de ataques bem-sucedidos. A seguir, tais recomendações são apresentadas:

Diretrizes de segurança de desenvolvimento. Para a construção da aplicação protetora é recomendado que sejam seguidas as mesmas diretrizes de codificação que são utilizadas para a SApp (Veja a **Atividade 1** disponível na Seção 4.1).

Manter as aplicações protetora e SApp separadas. Para evitar que a aplicação protetora tenha a sua segurança comprometida por algum ataque executado contra uma SApp, é importante que ambas sejam executadas em *hosts* diferentes. Nesse sentido, vale destacar que a comunicação entre o módulo `LibSelfProtection` e `AppMape` (veja a Figura 27) da abordagem proposta ocorre por meio de APIs REST, sendo necessário estabelecer meios de segurança como criptografia e autenticação para evitar que os dados sejam interceptados.

[11] MongoDB é um banco de dados NoSQL baseado em documentos

Compartimentalizar as funcionalidades dos módulos `AppControlPanel` e `AppMape`. Os módulos `AppMape` e `AppControlPanel` são manipulados por usuários que podem pertencer a três papéis, são eles: (i) **desenvolvimento**, onde engenheiros e arquitetos de software, e programadores definem no `AppControlPanel`, a aplicação a ser monitorada (SApp) e seus pontos de monitoramento; (ii) **especialista de segurança**, sendo este papel responsável por monitorar os eventos de segurança por meio do módulo `AppControlPanel` e acompanhar a performance do módulo `AppMape`, sendo que ambas atividades ocorrem na fase de *runtime* de uma SApp; e (iii) **mineração de dados e manutenção dos modelos de classificação**, onde especialistas em ciência de dados devem possuir acesso às interfaces de exportação de conjuntos de dados de comportamentos da aplicação (SApp) e atribuição dos modelos de classificação nos *loops* MAPE, sendo que ambas interfaces estão presentes no módulo `AppControlPanel`. Diante dos papéis supracitados, é importante que os usuários participantes das atividades estabelecidas na abordagem de autoproteção proposta neste trabalho tenham acesso específicos e restritos apenas às atividades que participam.

Criar redundância do repositório `MongoDB` e módulo `AppMape`. Para garantir alta disponibilidade dos módulos responsáveis pelo processo de autoproteção na fase de *runtime* é importante que os módulos `AppMape` e `AppControlPanel`, e o repositório de dados `MongoDB`, estejam disponíveis em mais de um nó de processamento. Assim, para diminuir os riscos de segurança, é recomendável que estes nós implementem diferentes plataformas, sistemas operacionais ou bibliotecas.

Seguir checklist de segurança para o repositório `MongoDB`. De acordo com Sahatqija *et al.* (2018), os sistemas de banco de dados NoSQL possuem algumas vulnerabilidades de segurança que podem comprometer a aplicação protetora. Portanto, recomenda-se que seja seguido o *checklist* de segurança proposto por MongoDB (2021). A seguir são apresentadas cada um dos itens do *checklist*:

- **Ativar controle de acesso e forçar autenticação**. Deve ser ativado um dos mecanismos de autenticação disponíveis pela comunidade do MongoDB.
- **Configurar o controle de acesso baseado em funções**. Para garantir que seja aplicado o princípio do menor privilégio devem ser criados usuários no repositório MongoDB, sendo um dedicado ao módulo `AppMape` e o outro ao `AppControlePanel`. Dessa forma, cada usuário deve ter acesso às funções e documentos específicos de cada módulo.
- **Criptografar a comunicação**. Deve ser ativado o protocolo TLS/SSL para todas as conexões de entrada e saída para garantir segurança na comunicação com os módulos `AppMape` e `AppControlPanel`.

- **Criptografar e proteger os dados**. Os dados armazenados devem ser criptografados utilizando o método nativo *WiredTiger* ou pelo sistema de arquivos do sistema operacional hospedeiro.
- **Limitar a exposição na rede**. Deve ser garantido que o ambiente de rede do *host* hospedeiro do repositório tenha configurado um *firewall*, grupos de segurança, além de restringir o acesso às portas e interfaces de rede.
- **Auditar a atividade do sistema**. Atividades de auditoria periódicas devem ser empregadas nas configurações do repositório, além das operações de acesso e eventos de conexão.
- **Executar o serviço com usuário dedicado**. O processo do repositório deve ser executado com usuário dedicado do sistema operacional hospedeiro e com permissões restritas a essa tarefa.
- **Executar o serviço com opções de configuração segura**. Devem ser desativados a execução de código *JavaScript* no processo do repositório.

4.3 Considerações finais

Neste capítulo foram apresentados os detalhes referentes à abordagem de autoproteção proposta neste trabalho. Na Seção 4.1 foram apresentados detalhes das características modeladas na abordagem, com base nos resultados obtidos no levantamento de estudos secundários e primários que compõem o Capítulo 3. Na sequência são apresentados os detalhes do processo e funcionamento da abordagem de autoproteção proposta neste trabalho. Na Seção 4.2 são apresentadas boas práticas e recomendações para o projeto de arquitetura, codificação e implantação dos módulos de autoproteção. Em resumo, o conteúdo deste capítulo fornece ao leitor uma visão mais completa e detalhada das características da abordagem de autoproteção, seus objetivos, ambiente de atuação proposto, seus atores e papéis em cada atividade. A seguir, no Capítulo 5, é apresentado um estudo de caso para avaliação da abordagem proposta.

5 ESTUDO DE CASO

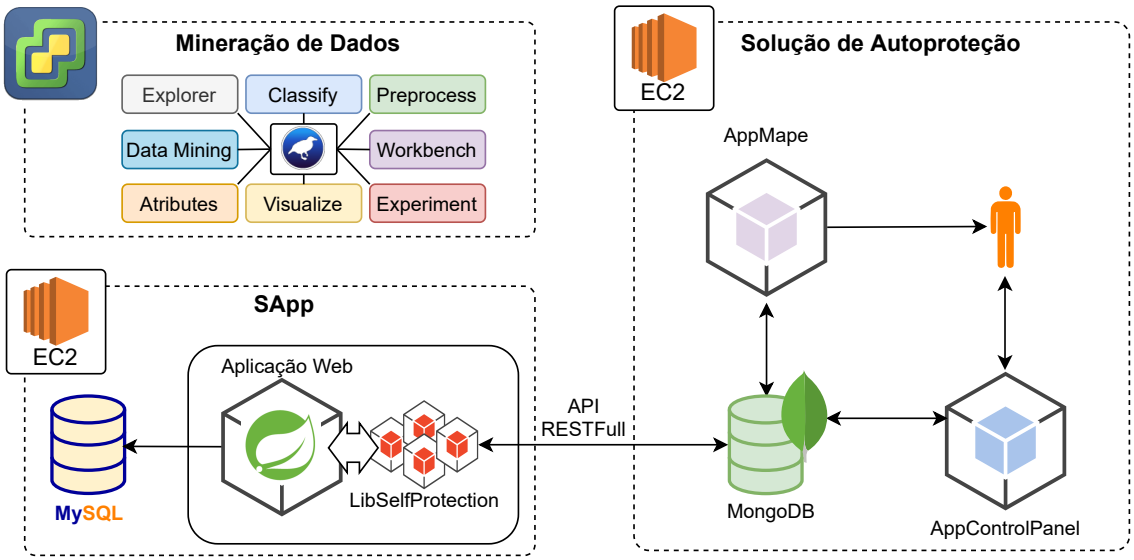
Este capítulo tem por objetivo apresentar uma visão detalhada de um estudo de caso que foi elaborado para testar e avaliar a viabilidade da abordagem de autoproteção proposta neste trabalho (veja o Capítulo 4). Para isso, o conteúdo deste capítulo foi organizado da seguinte maneira: na Seção 5.1 é apresentada uma visão geral do estudo de caso, onde são detalhados os componentes, as tecnologias e plataformas utilizadas; na Seção 5.2 são reportadas as fraquezas de segurança exploradas na SApp e como foram implementados/implantados os seus pontos de monitoramento; o processo e as ferramentas utilizados para a geração dos *detasets* de comportamentos são detalhados na Seção 5.3; na Seção 5.4 é apresentado o processo de implantação dos modelos de classificação nos *loops* MAPE; na Seção 5.5 é descrito o comportamento (ou seja, o modo de operar e monitorar) dos *loops* MAPE em tempo de execução; na Seção 5.6 é apresentado o processo de visualização dos dados resultantes dos *loops* MAPE; na Seção 5.7 são reportados os resultados dos testes aplicados no estudo de caso; A Seção 5.8 apresenta uma comparação da abordagem de autoproteção apresentada neste trabalho com as soluções extraídas do mapeamento da literatura, em que foram apresentadas no Capítulo 3; e, por fim, na Seção 5.9 são apresentadas as considerações finais do capítulo.

5.1 Visão geral e tecnologias

A Figura 41 ilustra os componentes, aplicações e ferramentas utilizados neste estudo de caso, os quais estão organizados em três ambientes de computação. No primeiro estão implantados o módulo `AppMape`, que é responsável por controlar os *loops* MAPE, o repositório de dados `MongoDB` (versão 4.4.6 *Community*) e o módulo `AppControlPanel`, que disponibiliza interfaces de gerenciamento da solução de autoproteção. No segundo, que foi denominado como "SApp", está implantada a SApp (ou seja, aplicação protegida) e seu banco de dados MySQL[1] (versão 5.6 *Community*). Ocupando o mesmo processo da SApp, está incorporado o módulo `LibSelfProtection`, que contém a implementação dos pontos de monitoramento da solução de autoproteção. Nomeado como "Mineração de Dados", o terceiro ambiente de computação contém o *workbench* Weka (versão 3.9.5), que é utilizada para os processos relacionados aos treinamentos de modelos de classificação. Em relação aos ambientes de computação, "Solução de Autoproteção" e "SApp" são instâncias do serviço EC2[2] do ambiente de computação em nuvem da organização Amazon AWS[3]. Em paralelo, o ambiente "Mineração de Dados" está

[1] https://www.mysql.com
[2] https://aws.amazon.com/pt/ec2
[3] https://aws.amazon.com

Figura 41 – Componentes



Fonte: Autoria própria

hospedado no Laboratório de Redes de Computadores e Sistemas Distribuídos do Instituto Federal de São Paulo - Campus Catanduva[4]. Cabe destacar que a implementação das aplicações `AppMape`, `AppControlPanel`, `LibSelfProtection` e a aplicação de domínio (SApp), foram implementadas por meio da linguagem de programação Java versão 1.8. A Tabela 4 apresenta as características relacionadas a cada um dos ambientes de computação, onde a coluna **S.O.** apresenta o sistema operacional, **S.V.** o sistema de virtualização hospedeiro, **RAM** a quantidade de memória RAM alocada e **CPU** a quantidade de núcleos de processamento alocados.

Tabela 4 – Características dos ambientes computacionais utilizados no estudo de caso

Ambiente	**S.O.**	**S.V**	**RAM**	**CPU**
Solução de Autoproteção	Amazon Linux v.2	Amazon AWS EC2	2GB	1 (vCPU)
SApp	Debian v.11	Amazon AWS EC2	2GB	1 (vCPU)
Mineração de Dados	Ubuntu v.18.04	VMWare VSphare v.5.5.0	32GB	8 núcleos

Fonte: Autoria própria

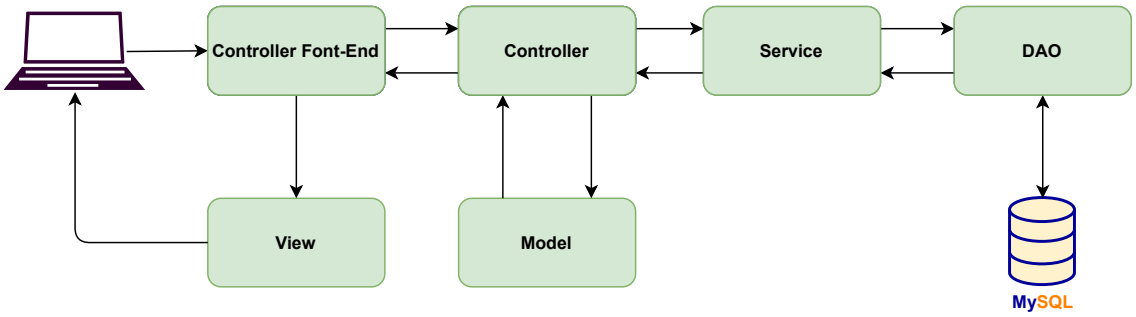
A SApp tem como objetivo principal obter parâmetros relacionados à sua aplicabilidade em uma situação de uso próxima a um cenário do mundo real. Nesse sentido, esse tipo de avaliação também possibilita determinar as eventuais limitações da abordagem de autoproteção, evidenciando pontos passíveis de melhoramento em trabalhos futuros. Para isso, foi desenvolvida uma aplicação *Web* para gerenciamento de empresas voltadas para o ramo de *Pet shop*, a qual será referenciada deste ponto em diante como `AppPetShop`. Em resumo, essa aplicação

[4] http://rsd.ctd.ifsp.edu.br

permite que pontos comerciais especializados em animais de estimação possam gerenciar os registros dos seus clientes e animais, além de controlar serviços de banho e tosa. Do ponto de vista arquitetural, a aplicação foi implementada sob a arquitetura MVC, onde existem camadas lógicas com responsabilidades específicas para lidar com a lógica da aplicação, apresentação da informação e manipulação das informações entre ambos. A implementação dessa aplicação teve como base o *framework* `SpringBoot`[5] versão 2.3.4, sendo instanciados os módulos `SpringWeb`, `SpringData JPA`, `SpringThymeleaf` e `Spring-Security`.

A Figura 42 ilustra as camadas que compõem a `AppPetShop`. A camada `Controller Front-End` é responsável por receber as requisições HTTPS que são enviadas por clientes (*Front-End*) e distribuí-las aos componentes `Controller` correspondentes. Em seguida, cada componente `Controller` extrai os dados e os parâmetros da URL direcionada à requisição HTTPS enviada pelo cliente. A partir dos dados extraídos, componentes `Service` são instanciados para executar as regras de negócio da aplicação `AppPetShop`. Nesse sentido, cada componente `Service` terceiriza aos componentes `DAOs` todas as ações relacionadas ao repositório de dados, como consultar, inserir, remover e alterar. Ao final da execução do componente `Controller`, objetos de domínio (`Model`) e/ou ações de controle resultantes são tratados e uma entidade da camada `View` (sendo uma página web) é encaminhada ao cliente que efetuou a requisição inicialmente.

Figura 42 – Arquitetura SApp



Fonte: Autoria própria

5.2 Pontos de monitoramento

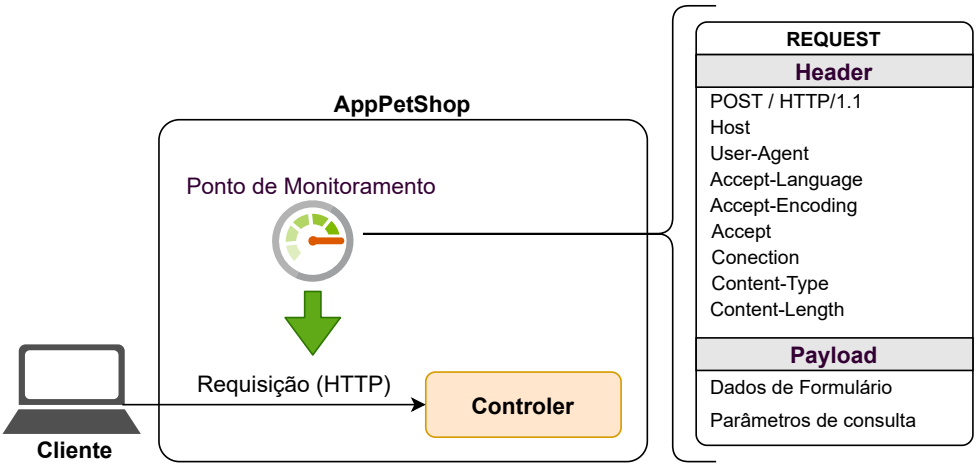
Inicialmente foram avaliadas as possíveis fraquezas e os eventuais pontos de vulnerabilidade existentes na aplicação `AppPetShop`, sendo identificado o ataque "injeção de SQL" como a principal ameaça. Embora o mesmo não seja novo, de acordo com o relatório "*OWASP Top 10 - 2017*"(OWASP, 2017b), este tipo de ataque ocupa a primeira posição dos mais recorrentes em aplicações *Web*. Tal informação motivou a adoção deste tipo de ataque para ser reportado em

[5] https://spring.io/

paralelo com a abordagem proposta neste trabalho. No que diz respeito aos tipos de ataques, vale mencionar que outros podem ser tratados em paralelo pela abordagem proposta. Basicamente, a opção por apresentar apenas um tipo de ataque neste estudo de caso está relacionada a dois fatores: (i) apresentação da abordagem em relação aos passos requeridos pela mesma; e (ii) atividades específicas que devem ser realizadas para cada tipo de ataque. Baseado no contexto exposto, para lidar com este tipo de ataque, técnicas de injeção de SQL foram estudadas, assim como a ferramenta *sqlmap*[6], cuja finalidade é automatizar esse tipo de ataque.

De acordo com OWASP (2021a), os ataques de injeção SQL exploram entradas de dados das aplicações *Web* que não possuem validação adequada. Nesse sentido, um usuário mal-intencionado pode injetar códigos SQL nas entradas de uma aplicação para falsificar sua identidade, adulterar dados existentes, consultar dados sensíveis, ou até mesmo interagir com o sistema operacional hospedeiro. Nessa direção vale destacar que a linguagem SQL permite diversas variações em sua estrutura e palavras chaves para os diferentes SGBDs (Sistema Gerenciador de Banco de Dados) disponíveis no mercado. Diante disso, é um grande desafio encontrar *datasets* públicos que permitam generalizar todas as variações de injeções possíveis. Assim, foram planejadas duas estratégias defensivas em profundidade para os pontos de monitoramento da `AppPetShop`. A primeira estratégia explora uma abordagem proativa (Figura 43), que intercepta as requisições HTTPS antes que cheguem aos componentes `Controller` da `AppPetShop`, onde são monitorados dados úteis (*paylod*) e dados de cabeçalho (*header*) que são enviados pelos clientes. A segunda estratégia (Figura 44) tenta identificar de modo reativo, sequências de execuções anômalas de comandos SQL, que podem ser originadas de ferramentas automatizadas para esse tipo de ataque. Diante do cenário exposto, caso alguma das estratégias falhe, a outra pode ser capaz de identificar uma ação maliciosa.

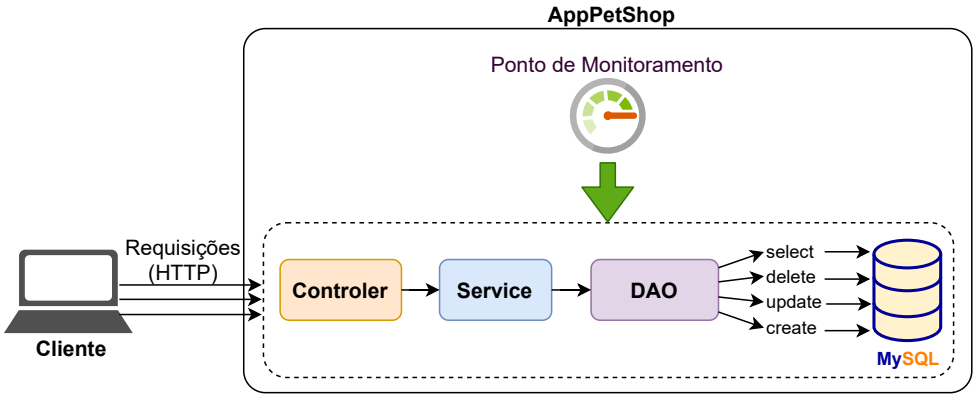
Figura 43 – Ponto de Monitoramento Proativo - Requisição HTTPS



Fonte: Autoria própria

[6] https://sqlmap.org

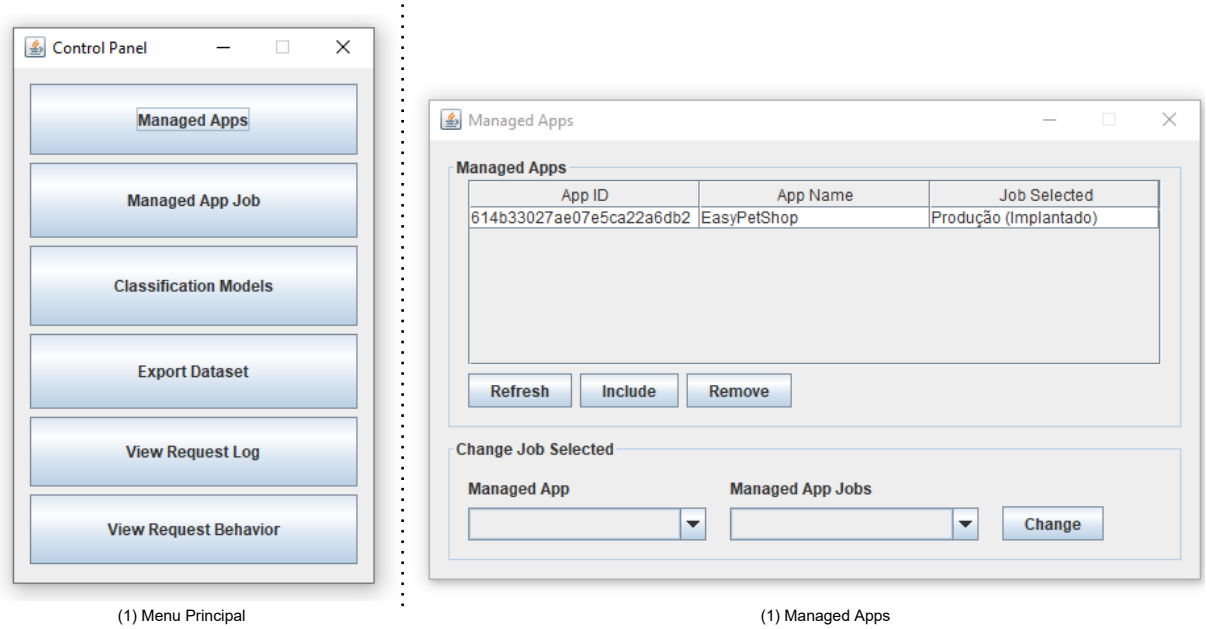
Figura 44 – Ponto de Monitoramento Reativo - Requisições HTTPS



Fonte: Autoria própria

Para criar os pontos de monitoramento, inicialmente os interessados (ou seja, engenheiro de software e/ou especialistas em ciência de dados/segurança) devem registrar a SApp na solução de autoproteção. Conforme ilustrado na Figura 45, uma SApp é registrada por meio do módulo `AppControlPanel`, onde deve ser informado o seu nome (coluna "App Name"). Quando o registro é efetuado, um código identificador para aplicação, chamado "App ID", é gerado automaticamente.

Figura 45 – AppControlPanel - Registro do `AppPetShop`



Fonte: Autoria própria

Como a aplicação `AppPetShop` foi implementada utilizando o *framework* SpringBoot, as configurações e parâmetros podem ser inseridos no arquivo `aplication.properties`. Neste sentido, conforme apresentado na Listagem 1, foram inseridos 5 tipos de configurações/parâmetros

para o `AppPetShop`, a saber são: (i) **spring.datasource.*** (Linhas 1 a 3), são propriedades que especificam o endereço, usuário e senha de conexão do banco de dados MySQL; (ii) **spring.jpa.*** (Linhas 5 a 7), são parâmetros para o mapeamento objeto-relacional do repositório MySQL; e (iii) **appmanaged.id** (Linha 9), onde é informado o valor da propriedade "App ID", que foi registrada no módulo `AppControlPanel` (veja a Figura 45), sendo essa propriedade responsável por vincular a SApp à solução de autoproteção.

Listagem 1 – Arquivo "application.properties"

```
1 spring.datasource.url=jdbc:mysql://localhost:3306/petshop?
2 spring.datasource.username=root
3 spring.datasource.password=123456
4
5 spring.jpa.hibernate.ddl-auto=update
6 spring.jpa.properties.hibernate.ejb.interceptor=
      ↪ com.sistemas.armartins.interceptors.HibernateInterceptor
7 spring.jpa.open-in-view=true
8
9 appmanaged.id=614b33027ae07e5ca22a6db2
```

Conforme ilustrado na Listagem 2, para inserir o ponto de monitoramento proativo à SApp, foi criado um componente chamado `TransactionFilter` que herda funcionalidades da classe `javax.servlet.Filter` (Linha 3). Basicamente, esse componente intercepta dinamicamente todas as requisições HTTPS antes que o fluxo de execução chegue aos componentes `Controller`. Conforme pode ser observado nas Linhas 4 e 5, por meio da técnica de injeção de código, a propriedade `appmanaged.id`, que foi definida no arquivo `application.properties` (veja Listagem 1), é recuperada e inserida na variável `appID`. Assim, a cada requisição HTTPS que é interceptada pela classe `TransactionFilter`, um objeto `mpr` é instanciado (Linha 16) a partir da classe `MonitoringPointRequest` (veja Figura 30 – Capítulo 4) do módulo `LibSelfProtection` (veja Figura 27 – Capítulo 4). Na sequência, a requisição, encapsulada em um objeto do tipo `ContentCachingRequestWrappert` (Linha 12), e o conteúdo da variável `appID` são enviados (Linha 17) para o `loop` MAPE.

Listagem 2 – Ponto de Monitoramento Proativo

```
1  @Component
2  @Order(1)
3  public class TransactionFilter implements Filter {
4      @Value("${appmanaged.id}")
5      private String appID;
6
7      @Override
8      public void doFilter(ServletRequest request, ServletResponse response,
           ↪ FilterChain chain)
9      throws IOException, ServletException {
10         try {
```

```
11            HttpServletRequest currentRequest = (HttpServletRequest) request;
12            ContentCachingRequestWrapper wrappedRequest = new
                  ↪ ContentCachingRequestWrapper(currentRequest);
13
14            chain.doFilter(wrappedRequest, response);
15
16            MonitoringPointRequest mpr = MonitoringPointRequest.getInstance();
17            mpr.run(wrappedRequest, appID);
18         } catch (URISyntaxException ex) {
19            java.util.logging.Logger.getLogger(TransactionFilter.class.getName())
20            .log(Level.SEVERE, null, ex);
21         } catch (Exception ex) {
22             java.util.logging.Logger.getLogger(TransactionFilter.class.getName())
23             .log(Level.SEVERE, null, ex);
24         }
25      }
26   }
```

O ponto de monitoramento reativo é apresentado na Listagem 3, onde é definido o componente `RequestInterceptor`, que implementa a interface `HandlerInterceptor`[7] (Linha 2). Em resumo, essa interface permite interceptar os fluxos de execuções enviados aos componentes `Controller`. Em paralelo, por meio de injeção do código, a variável `appID` recupera o valor da propriedade `appmanaged.id` (Linhas 6 e 7), que foi definido no arquivo `application.properties`. Na Linha 18 é recuperado o nome do componente `Controller` que é alvo da requisição HTTPS, que é armazenado na variável `controller` (Linha 11). Na Linha 26, o objeto `mpc`, instância da classe `MonitoringPointController` (veja Figura 30 – Capítulo 4), é responsável por enviar os dados para o loop MAPE (Linha 28). Cabe destacar a Linha 27, que recupera os comandos SQL enviados ao repositório MySQL por meio do objeto `hibernateInterceptor`[8]

Listagem 3 – Ponto de Monitoramento Reativo

```
1  @Component
2  public class RequestInterceptor implements HandlerInterceptor{
3     @Autowired
4     HibernateInterceptor hibernateInterceptor;
5
6     @Value("${appmanaged.id}")
7     private String appID;
8
9     private static final Logger LOGGER =
          ↪ LoggerFactory.getLogger(HibernateInterceptor.class);
10
11    private String controller;
```

[7] org.springframework.web.servlet.HandlerInterceptor

[8] O objeto `hibernateInterceptor` é instância da classe `HibernateIntercetor` na Linha 4, que herda atributos e métodos da classe `org.hibernate.EmptyInterceptor`.

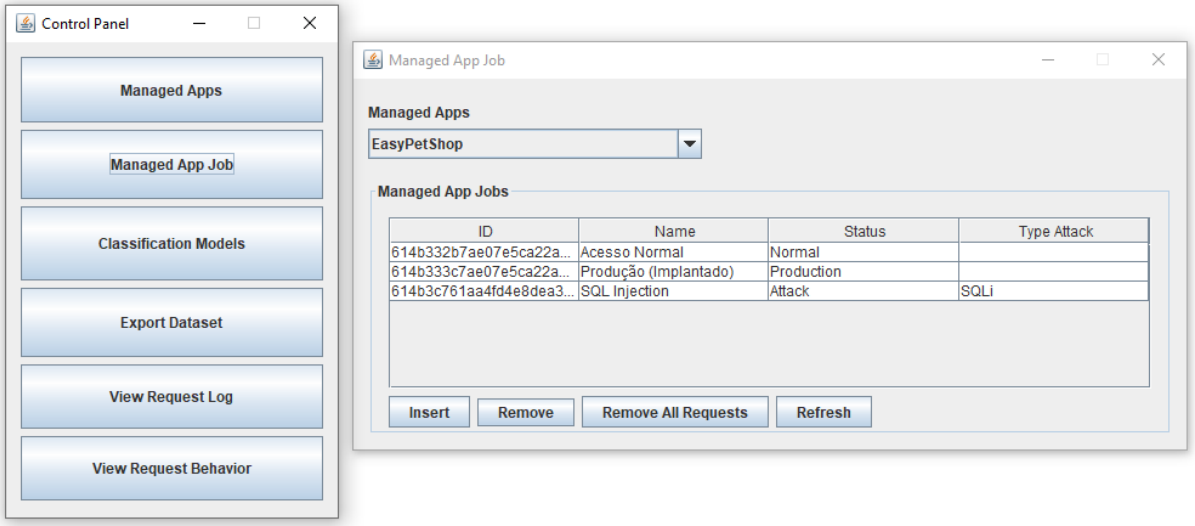
```
12
13      @Override
14      public boolean preHandle(HttpServletRequest request, HttpServletResponse
            ↪ response, Object handler) throws Exception {
15          hibernateInterceptor.startCounter();
16          if (handler instanceof HandlerMethod){
17              HandlerMethod method = (HandlerMethod) handler;
18              controller = method.getBean().getClass().getSimpleName();
19          }
20          return HandlerInterceptor.super.preHandle(request, response, handler);
21      }
22
23      @Override
24      public void afterCompletion(HttpServletRequest request, HttpServletResponse
            ↪ arg1, Object handler, Exception arg3) throws Exception {
25          if (handler instanceof HandlerMethod){
26              MonitoringPointController mpc = new MonitoringPointController();
27              List<String> sqls = hibernateInterceptor.getQuery();
28              mpc.run(appID, sqls, request, controller);
29          }
30          hibernateInterceptor.clearCounter();
31      }
32  }
```

5.3 Criação e exportação dos *datasets* de *features*

Este processo iniciou quando a primeira versão do `AppPetShop` estava funcional e implantada no ambiente de testes. Basicamente, este processo é composto por três atividades, a saber: (i) **formação de *datasets* de comportamentos normais**, onde requisições normais foram enviadas para a aplicação `AppPetShop` por meio de ações manuais via navegadores *Web*; (ii) **formação de *datasets* de comportamentos maliciosos**, onde a partir de ferramentas automatizadas de teste de penetração e *scripts* foram enviadas requisições maliciosas para a `AppPetShop`; e (iii) **exportação dos *datasets* de `features`**, onde a partir dos *datasets* de comportamentos, ocorre o processo de extração de *features*, sendo gerado ao final os *datasets* de *features*. As atividades de formação de *datasets* de comportamentos são representadas no módulo `AppControlPanel` como `ManagedAppJob`, possuindo as variações "Normal" para acessos normais e "Attack" para os maliciosos, como ilustra a Figura 46. As variações de `ManagedAppJob` são "setadas" a cada uma das atividades por meio da interface *Managed Apps* (veja Figura 45 – lado direito). Cabe destacar que as requisições maliciosas enviadas para a `AppPetShop` foram realizadas por meio da ferramenta *sqlmap* versão 1.5.12 com auxílio da ferramenta *BurpSuite Community Edition* versão 2021.8.4[9].

[9] https://portswigger.net/burp

Figura 46 – AppControlPanel - Atribuição do `AppPetShop`

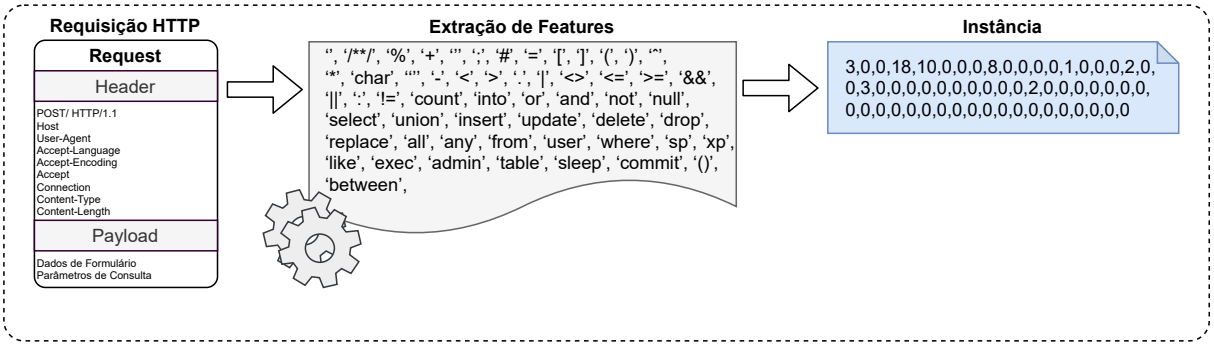


Fonte: Autoria própria

Em relação ao processo de extração de *features* para defesa proativa, a partir de cada requisição HTTP, que é constituída pelas seções *header* e *payload* (veja o componente **`Request`** – Figura 47), as *features* são extraídas por meio da contabilização de palavras chaves e caracteres específicos (veja o componente "**Extração de *features***" – Figura 47) que são recorrentes em injeções de código SQL. Ao final do processo de extração é gerada uma instância que irá compor o *dataset* de *features* (veja o componente "**Instância**" – Figura 47). Em paralelo, conforme ilustrado na Figura 48, as *features* extraídas para o mecanismo de defesa reativo são formadas a partir de um conjunto de requisições que são capturadas em um intervalo de tempo em 10 segundos. Em cada requisição são contabilizadas, a partir de cada componente **`Controller`**, a quantidade de requisições efetuadas para cada método HTTP (veja componente "**Requisições HTTP**" – Figura 48) e a média de *bytes* referente as consultas SQL (veja componente "**Consultas SQL**" – Figura 48) enviadas ao banco de dados MySQL. No final é gerada uma instância que irá compor o *dataset* de *features* (veja componente "**Instância**" – Figura 48). A Tabela 5 apresenta uma breve descrição para as *features* utilizadas.

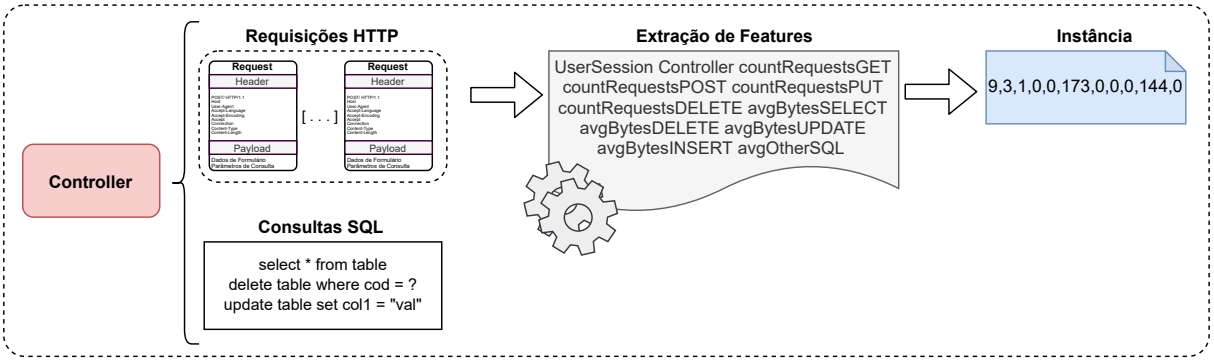
Por meio da ferramenta *sqlmap*, foi possível formar um *dataset* capaz de generalizar diferentes variações de ataques de injeção de SQL para o banco de dados MySQL. Em seguida foi iniciado o processo de extração de *features* e exportação dos *datasets* de *features*, onde o processo ocorre por meio da interface `Exportação de Datasets`, que é ilustrado na Figura 49. Essa interface permite extrair as *features* a partir das requisições enviadas para a aplicação `AppPetShop`, assim como para os *datasets* públicos como o "*HTTP CSIC Torpeda 2012*" e o "*ECML/PKDD 2007*". Assim, ao final são gerados dois arquivos em formato `CSV`, sendo que ambos representam *datasets* de *features* para os *loops* que lidam com o ataque de maneira

Figura 47 – Extração de *features* para defesa proativa



Fonte: Autoria própria

Figura 48 – Extração de *features* para defesa reativa



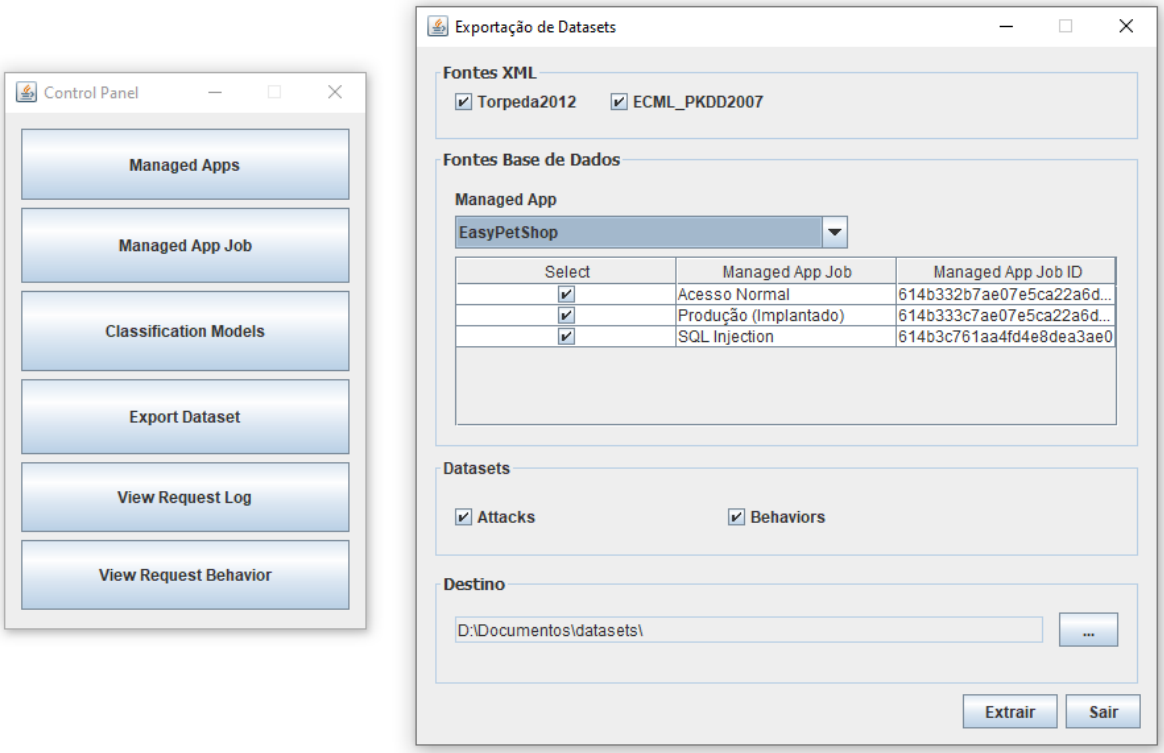
Fonte: Autoria própria

Tabela 5 – *Features* para detecção proativa

Features	**Descrição**
`UserSession`	Conjunto de caracteres que representam a sessão estabelecida entre o cliente e a `AppPetShop`
`Controller`	Identificador do componente `Controller` responsável por atender a requisição HTTPS enviada pelo cliente
`countRequestsGET`	Quantidade de requisições enviadas por meio do método `GET`
`countRequestsPOST`	Quantidade de requisições enviadas por meio do método `POST`
`countRequestsPUT`	Quantidade de requisições enviadas por meio do método `PUT`
`countRequestsDELETE`	Quantidade de requisições enviadas por meio do método `DELETE`
`avgBytesSELECT`	Média de *bytes* de consultas SQL enviadas ao banco de dados
`avgBytesDELETE`	Média de *bytes* de comandos `DELETE` enviadas ao banco de dados
`avgBytesUPDATE`	Média de *bytes* de comandos `UPDATE` enviadas ao banco de dados
`avgBytesINSERT`	Média de *bytes* de comandos `INSERT` enviadas ao banco de dados
`avgOtherSQL`	Média de *bytes* de comandos que não sejam de consulta, `DELETE`, `UPDATE`, e `INSERT` que são enviados ao banco de dados

Fonte: Autoria própria

Figura 49 – AppControlPanel - Exportação dos *datasets*



Fonte: Autoria própria

proativa e reativa, respectivamente.

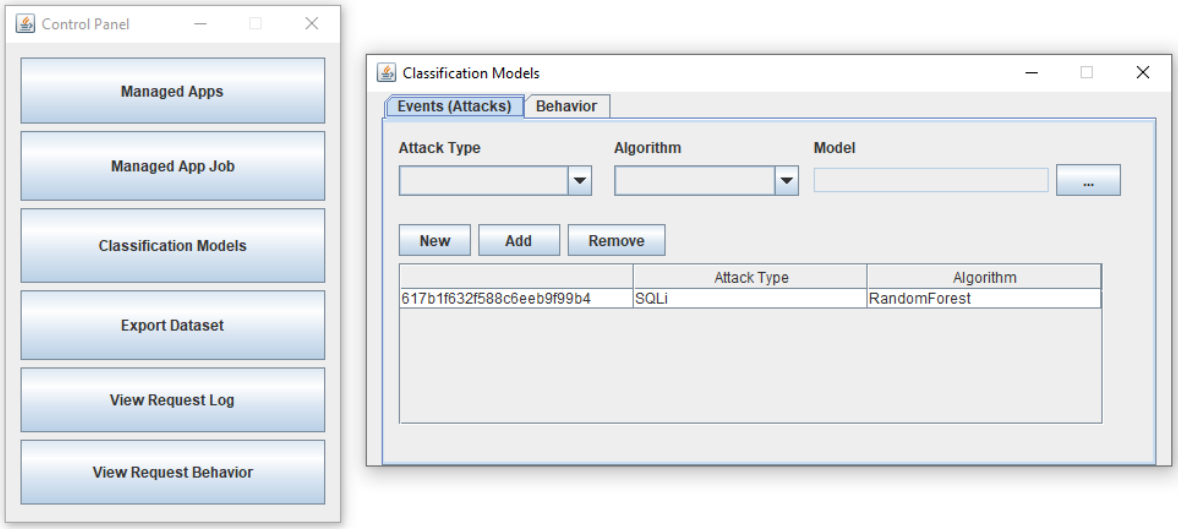
5.4 Implantação dos modelos de classificação

O processo de mineração de dados nasce a partir dos *datasets* de *features* exportados pelo módulo `AppControlPanel`. Na sequência, os *datasets* são carregados no *workbench* Weka, onde são realizados os processos de preparação dos dados, seleção de atributos, balanceamento das classes de dados e avaliação dos algoritmos de classificação. Ao final, o modelo de classificação é treinado e serializado em um arquivo para ser incorporado ao *loop* MAPE que lida com o ataque "injeção de SQL" correspondente. O Apêndice C apresenta em detalhes as atividades executadas no *workbench* Weka para a criação dos modelos de classificação proativo e reativo.

A Figura 50 ilustra a interface `Classification Models` do módulo `AppControlPanel`, que é responsável por incorporar os modelo de treinamento aos *loops* MAPE. Sobre essa operação, vale destacar que quando os modelos são selecionados e salvos por meio da interface, eles são armazenados no repositório `MongoDB`, onde é utilizada a especificação GridFS[10] que permite armazenar arquivos com tamanho acima de 16MBytes. Em paralelo, quando os *loops* entram em

[10] https://docs.mongodb.com/manual/core/gridfs

Figura 50 – AppControlPanel - Modelos de Classificação



Fonte: Autoria própria

execução por meio do módulo `AppMape`, os modelos são recuperados do repositório `MongoDB` e são carregados em memória para funcionamento da aplicação protetora.

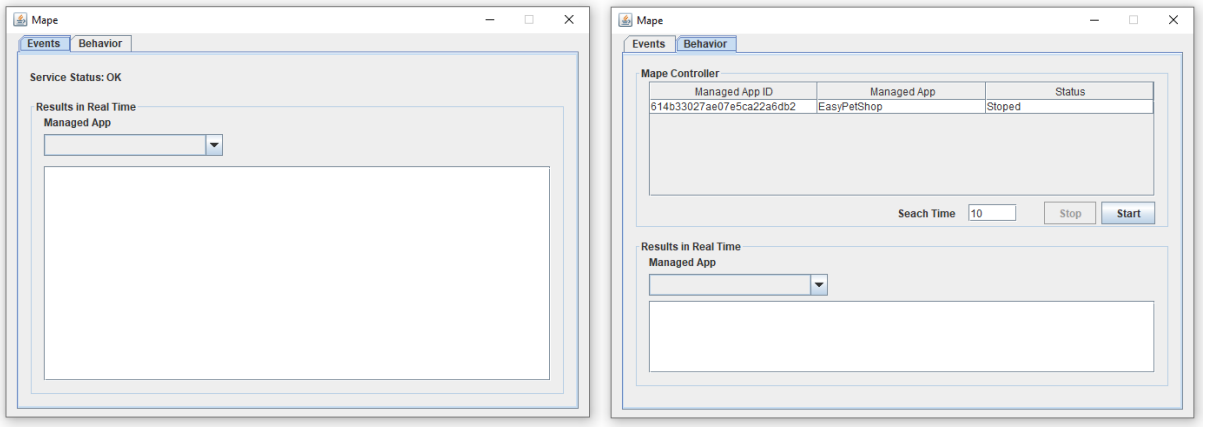
No que diz respeito a seleção dos modelos de treinamento, a interface `Classification Models` é organizada em duas abas (veja Figura 50). Na primeira, chamada de `Events (Attacks)`, é possível incluir modelos de classificação para os pontos de monitoramento proativos. Embora neste estudo de caso tenha sido desenvolvido apenas um modelo para detecção de "injeção de SQL" para fins de avaliação da abordagem proposta neste trabalho, a interface permite que sejam incluídos modelos para outros tipos de ataques também. Na segunda, que é chamada de `Behavior`, é possível incluir o modelo de classificação destinado aos pontos de monitoramento reativos (veja Seção 5.2).

5.5 Execução dos *loops* MAPE

Na Figura 51 é ilustrado o módulo `AppMape` em execução, onde sua interface é composta por duas abas principais. A primeira, chamada de `Events`, representa o *loop* MAPE responsável pelo ponto de monitoramento proativo. A segunda, que recebe o nome de `Behavior`, representa o *loop* MAPE responsável pelo ponto de monitoramento reativo (veja Seção 5.2).

Ao executar o módulo `AppMape`, é iniciado automaticamente um serviço *Web* que disponibiliza métodos RESTfull na porta 8080. Estes métodos são consumidos pela aplicação `AppPetShop` por meio do módulo `LibSelfProtection`. Dessa forma, a cada requisição HTTP/HTTPS recebida pela aplicação, dados são extraídos da requisição e enviados ao método

Figura 51 – AppMape



Fonte: Autoria própria

RESTfull. Esses dados recebidos pelos métodos alimentam o *loop* MAPE, que inicia seu ciclo de autoproteção. Em paralelo, a aba `Events` possui uma janela chamada `Results in real Time`, que permite visualizar por meio de *logs* cada uma das requisições recebidas em tempo real.

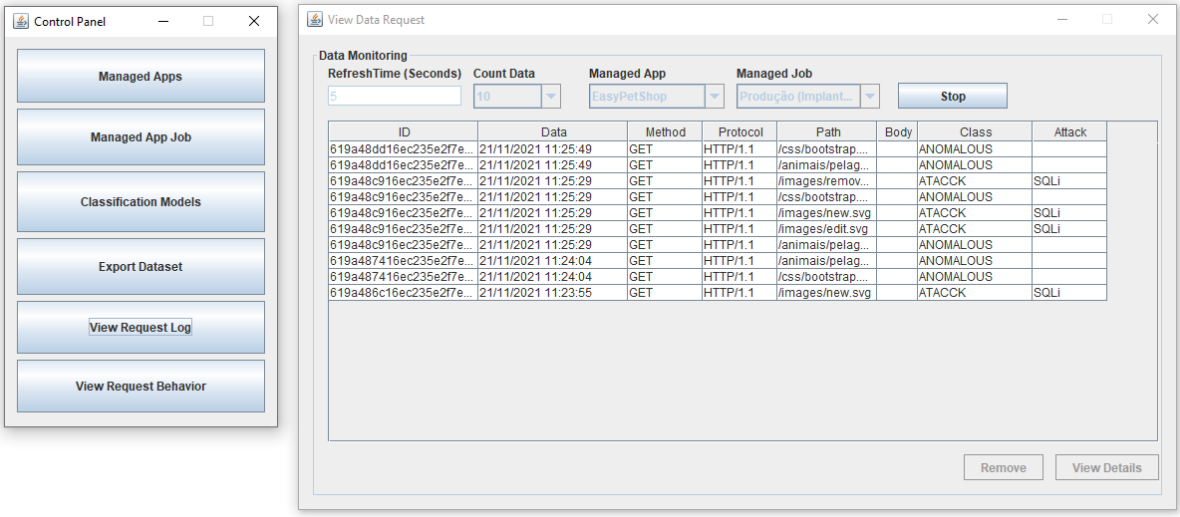
Conforme ilustrado na Figura 51, na aba `Behavior`, contida no módulo `AppMape`, é iniciado o *loop* MAPE que monitora os dados gerados pelos pontos de monitoramento reativos (veja Seção 5.2). Para isso, os interessados devem informar um valor numérico que representa uma unidade de tempo em segundos que será utilizada como referência entre as buscas dos dados de monitoramento que estão armazenadas no repositório `MongoDB`. Em outras palavras, por meio de um laço de repetição, os dados referentes às requisições HTTP/HTTPS disparadas a aplicação `AppPetShop` são coletadas do repositório `MongoDB` e alimentam o *loop* MAPE. Em paralelo, a mesma interface permite que o tratamento das requisições coletadas sejam acompanhadas em tempo real por meio *logs* disponíveis na janela `Results in Real Time`.

5.6 Visualização dos dados

As Figuras 52 e 53 ilustram, respectivamente, as interfaces `View Data Request` e `View Data Behavior`, que estão disponíveis no módulo `AppControlPanel`. Em resumo, essas interfaces permitem monitorar a evolução e performance dos *loops* MAPE da aplicação protetora que estão disponíveis no módulo `AppMape`. Ambas as interfaces buscam dados de *logs* referentes à execução dos *loops* que estão armazenados no repositório `MongoDB`. Para isso, os interessados que estão manipulando essas interfaces devem informar um intervalo de tempo (ou seja, valor numérico) em segundos para que as buscas nos repositórios sejam realizadas. Além disso, em ambas as interfaces também é necessário informar um conjunto de dados de entrada para permitir que as buscas sejam efetuadas, a saber: (i) `Refresh Time (Seconds)`, que define uma

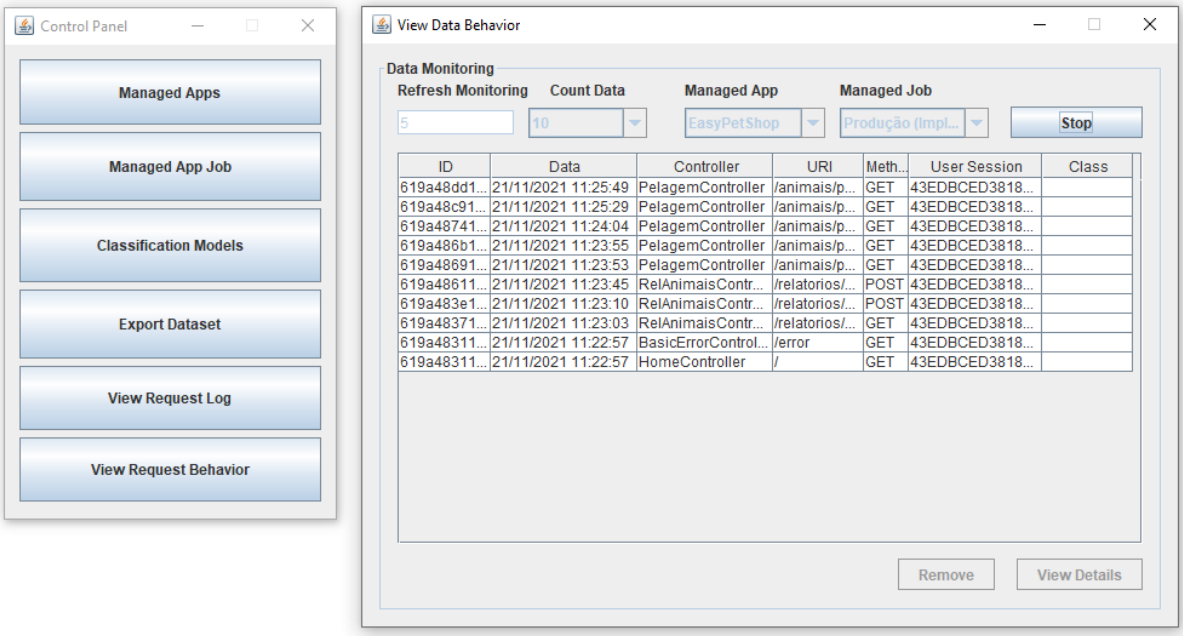
unidade de tempo em segundos, que serve como referência entre as buscas; (ii) `Count Data`, que define a quantidade de requisições processadas que serão exibidas. Em outras palavras, serão selecionados dados das últimas requisições processadas; e (iii) `ManagedApp` e `ManagedJob`, onde é selecionado o SApp que será monitorado.

Figura 52 – AppControlPanel - Visualização de logs de requisições



Fonte: Autoria própria

Figura 53 – AppControlPanel - Visualização de logs de comportamento



Fonte: Autoria própria

Na Figura 52, que ilustra a interface `View Request Log`, é possível acompanhar a evolução e performance do `loop` MAPE que é responsável pela defesa proativa da aplicação alvo

(ou seja, `AppPetShop`). Logo, os dados que são obtidos nas buscas ao repositório `MongoDB` são exibidos em uma tabela, onde são organizados nas seguintes colunas: (i) `ID`, que apresenta o código identificador do registro no repositório; (ii) `Data`, que representa data e hora da requisição HTTP/HTTPS; (iii) `Method`, apresenta o método HTTP utilizado para fazer a requisição; (iv) `Protocol`, apresenta a versão do protocolo HTTP utilizado para fazer a requisição; (v) `Path`, apresenta o endereço URI (do inglês, *Uniform Resource Identifier*) do recurso requisitado; (vi) `Body`, os dados da carga útil da requisição; e, por fim (vii) `Attack`, o resultado da classificação processado pelo *loop* MAPE.

Na Figura 53 é ilustrada a execução da interface `Request Behavior`, cujo objetivo é monitorar a evolução e performance do *loop* MAPE responsável pela defesa reativa da aplicação alvo (ou seja, `AppPetShop`). Em paralelo, os dados que são obtidos a partir das buscas, são exibidos na tabela disponível na interface, onde são organizados em colunas da seguinte maneira: (i) `ID`, que apresenta o código identificador do registro no repositório; (ii) `Data`, onde é apresentado a data e hora da requisição; (iii) `Controller`, que apresenta o nome do componente `Controller` requisitado; (iv) `URI`, que apresenta a URI do recurso requisitado; (v) `Method`, onde é apresentado o método HTTP utilizado para requisição do recurso; (vi) `UserSession`, onde é apresentado o código identificador da sessão do cliente que requisitou o recurso; e, por fim, (vii) `Class`, que apresenta do resultado da classificação obtida pelo *loop* MAPE.

5.7 Testes e resultados

Esta seção tem por objetivo apresentar os detalhes sobre o processo de teste realizado durante o desenvolvimento deste estudo de caso, além de apresentar os resultados que confirmem a viabilidade da abordagem de autoproteção proposta neste trabalho. Como os testes foram realizados em três etapas, nas Seções 5.7.1, 5.7.2 e 5.7.3 são descritas como cada etapa foi conduzida. Por fim, na Seção 5.7.1 são reportados e discutido os resultados em cada etapa de teste.

5.7.1 Etapa 1 de testes

Esta etapa foi realizada em duas fases, sendo a primeira relacionada aos processos relacionados a geração de *datasets* de *features* e a segunda voltada para a criação dos modelos de classificação via *workbench* Weka, que são utilizados para detectar as ações maliciosas.

A fase de geração de *datasets* de *features* é iniciada quando os componentes que compõem o estudo de caso (veja Figura 41) estão devidamente implantados e funcionais. Neste contexto, inicialmente foi conduzido uma sequência de testes funcionais de modo manual em todas as funcionalidades da aplicação `AppPetShop`, o que permitiu a geração de requisições normais à

aplicação. Na sequência, por meio da ferramenta *sqlmap*, foram realizados testes de penetração nas funcionalidades disponíveis na aplicação `AppPetShop`, onde foi possível gerar requisições de ataque. Ao final dos testes funcionais e de penetração, por meio do módulo `AppControlPanel`, foram gerados os arquivos de *datasets* em formato `csv`. Retomando o tipo de ataque que está sendo explorado neste estudo de caso, o arquivo "`sqli.csv`" deve ser destinado para a criação do modelo de classificação para o *loop* MAPE proativo, assim como o arquivo "`behavior.csv`" deve ser destinado ao modelo de classificação para criação do *loop* MAPE reativo.

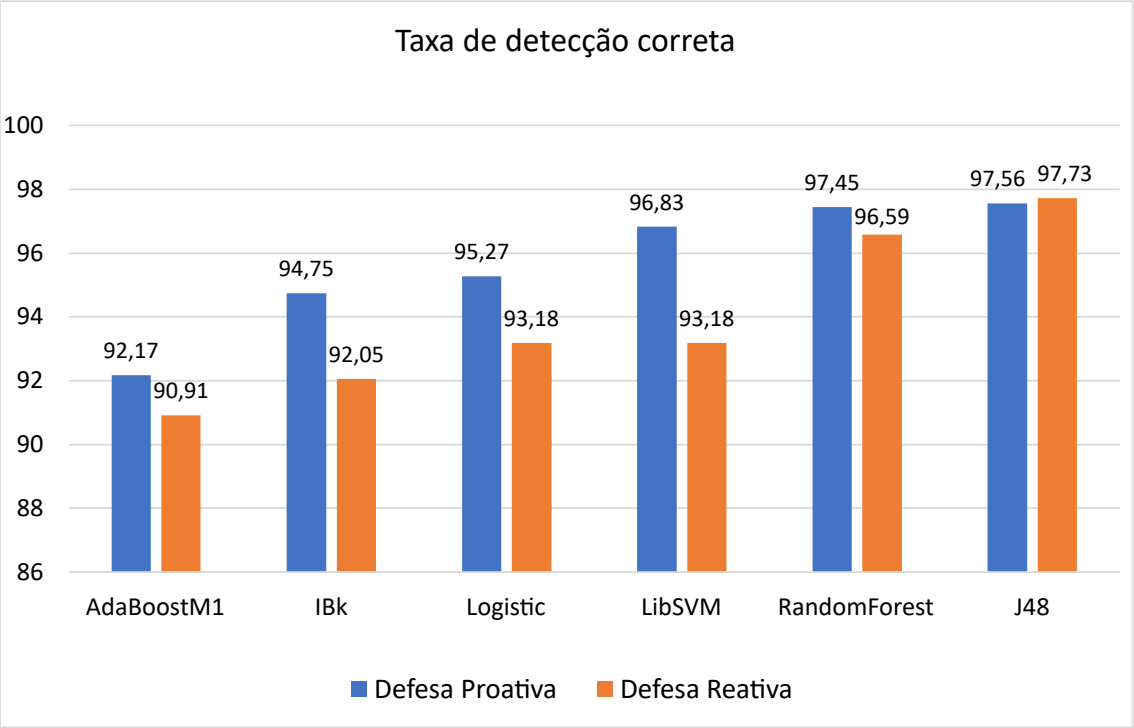
No que diz respeito ao arquivo "`sqli.csv`" foram gerados 65.503 instâncias rotuladas como normais e 42.287 instâncias rotuladas como ataques. Nessa direção, o arquivo "`behavior.csv`" é composto por 63 instâncias rotuladas como normais e 80 são rotuladas como ataques. Cabe ressaltar que o arquivo "`sqli.csv`" foi gerado a partir dos *datasets* **HTTP CSIC Torpeda 2012** e **ECML/PKDD 2007**, enquanto o arquivo "`behavior.csv`" foi gerado a partir dos dados armazenados no repositório `MongoDB`, que foram gerados a partir dos testes funcionais e de penetração.

Após a geração dos *datasets* de *features*, a fase de criação dos modelos de classificação foi iniciada. Para cada arquivo (sendo "`sql.csv`" para a detecção proativa e "`behavior.csv`" para a detecção reativa) foram aplicados filtros aos dados e na sequência também foi avaliada a possibilidade de baleamento de classes e seleção de atributos. Em seguida, foram avaliados seis algoritmos de classificação em relação aos modelos de detecção proativo e reativo e suas respectivas taxas de detecção correta, como ilustra a Figura 54. As Figuras 55 e 56 ilustram as curvas ROC relacionadas a cada algoritmo avaliado para os modelos de detecção proativo e reativo, respectivamente. Em relação aos algoritmos, **AdaBoostM1** é um classificador para problemas de classe nominal, **J48** é um classificador baseado em árvore, **Logistic** é um algoritmo de regressão para classificação binária, **LibSVM** é uma biblioteca para *Support Vector Machines*, e **IBk** (*K-Nearest Neighbor*) e **Random Forest** são algoritmos para lidar com a classificação e problemas de regressão. Embora o algoritmo **J48** tenha apresentado melhor taxa de detecção correta (veja Figura 54) para ambos os modelos (proativo e reativo), o algoritmo **Random Forest** foi selecionado para ambos os modelos de classificação. Essa escolha teve como base a análise das curvas ROC de cada algoritmo, onde pode-se observar o funcionamento de cada um deles em relação aos acertos e erros dos modelos.

5.7.2 Etapa 2 de testes

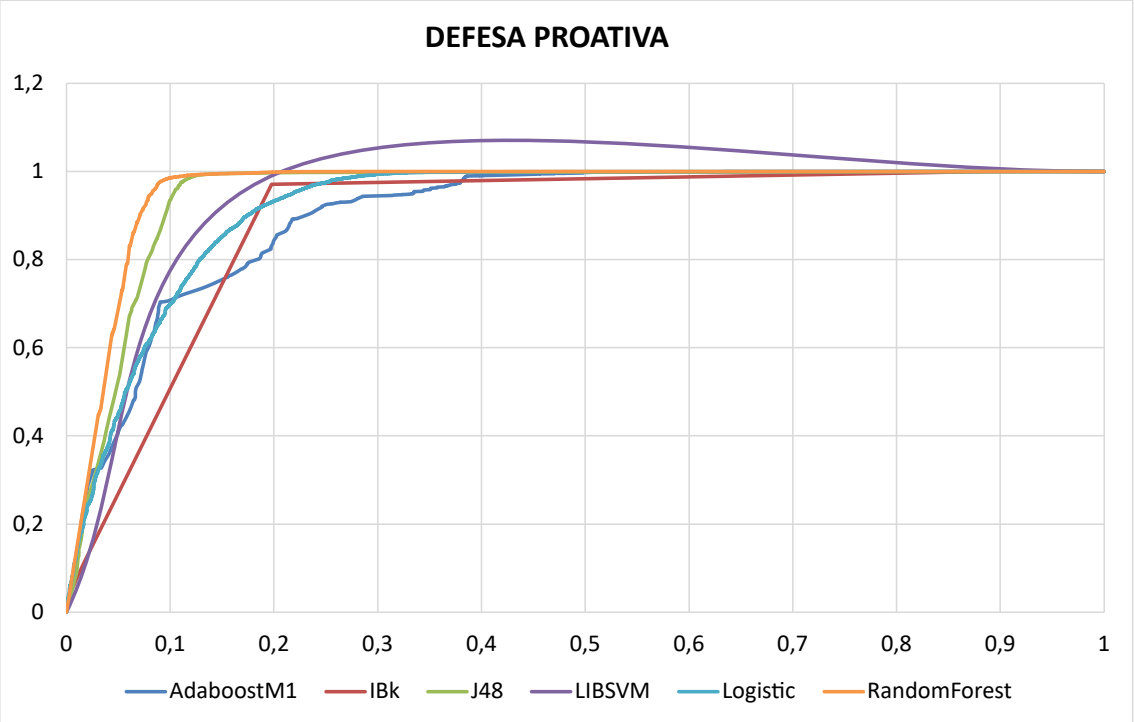
Esta etapa tem por objetivo avaliar a eficácia da abordagem de autoproteção e seus componentes em um ambiente computacional que simula um cenário real de execução. Os modelos de classificação gerados na *Etapa 1* (veja Seção 5.7.1) foram adicionados aos *loops* MAPE por meio do módulo `AppControlPanel` (veja Seção 5.4). Na sequência, os mesmos

Figura 54 – Porcentagens de Detecções Corretas - PDCs



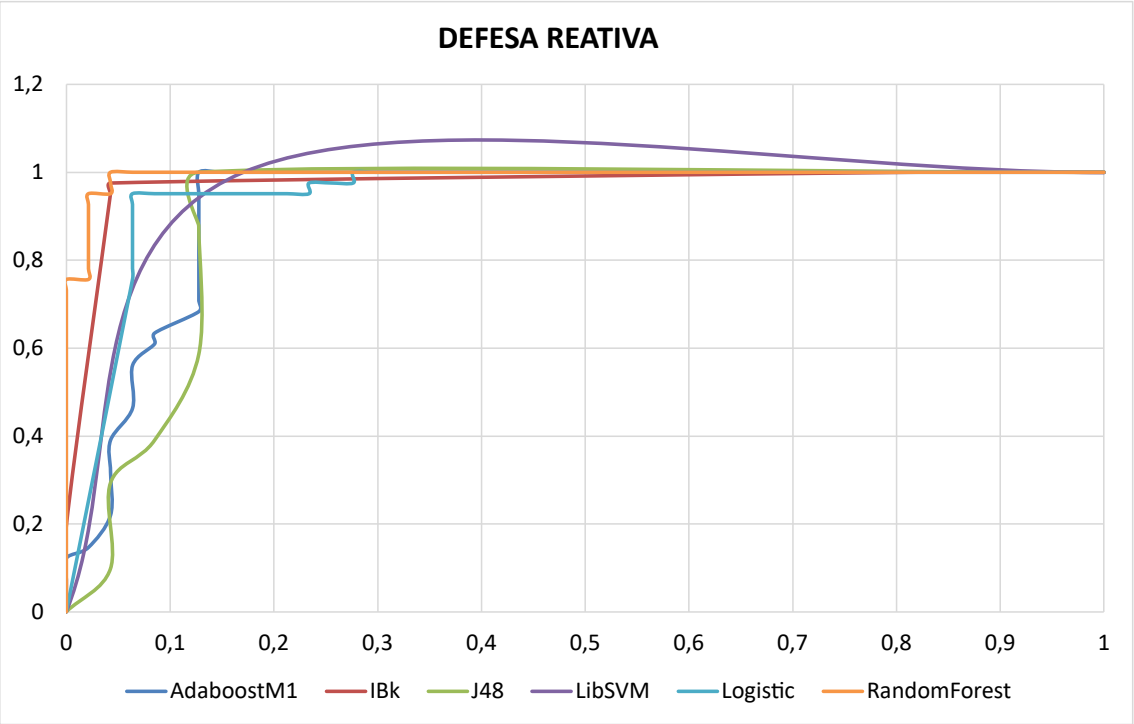
Fonte: Autoria própria

Figura 55 – Curva ROC - Detecção Proativo



Fonte: Autoria própria

Figura 56 – Curva ROC - Detecção Reativo



Fonte: Autoria própria

testes funcionais e de penetração que foram realizados na aplicação `AppPetShop` na **Etapa 1**, foram repetidos para verificar se o módulo `AppMape` é capaz de identificar acessos normais e/ou anômalos. Sobre essa atividade, ambos os *loops* MAPE foram capazes de identificar as requisições normais e os ataques maliciosos. Durante a execução desta etapa observou-se que as requisições direcionadas ao endereço "`/css/bootstrap.min.css`" foram classificadas erroneamente como ataque. Em seguida, os dados contidos no *dataset* de *features* foi reavaliado e a causa identificada para tal anomalia foi a presença de dois caracteres "." (ponto) na URL, o que não ocorre em nenhuma das instâncias do *dataset*.

Como forma de contornar a anomalia do modelo de classificação supracitado, um conjunto de requisições normais foi executado para a aplicação `AppPetShop`. Como resultado, 305 novas instâncias adicionadas ao *dataset* "`sqli.csv`", que foi gerado inicialmente na **Etapa 1** de testes (veja Seção 5.7.1). Em seguida, a fase de criação do modelo de classificação executada na **Etapa 1** foi refeita e um novo modelo de classificação foi gerado e incorporado ao *loop* de controle proativo. Na sequência, novos testes funcionais e de penetração foram executados e geraram resultados positivos, onde as requisições direcionadas ao endereço "`/css/bootstrap.min.css`" passaram a ser classificadas como normal.

5.7.3 Etapa 3 de testes

Nesta etapa os testes de penetração foram refeitos com apoio da ferramenta `jSQL Injection`[11] para avaliar a capacidade de defesa em profundidade da abordagem de autoproteção proposta neste trabalho. De acordo com Mishra e Kumar (2020), a ferramenta `jSQL` possui melhor performance em relação ao *sqlmap* por ser mais rápido em efetuar varreduras em busca de vulnerabilidades, porém o *sqlmap* possui um repertório maior de testes que permite extrair "impressão digital" do banco de dados. Neste sentido, é esperado que o *dataset* "`behavior.csv`" não seja capaz de generalizar e identificar ataques para esse tipo de ferramenta. Em contrapartida, espera-se que o `sqli.csv` possa generalizar e identificar ataques a partir dessa ferramenta.

Ao fazer os testes de penetração com a ferramenta `jSQL` foi possível confirmar a hipótese supracitada, onde a solução de autoproteção foi capaz de identificar proativamente os ataques de injeção, enquanto o *loop* de defesa reativa não foi possível identificar tais ataques. Neste sentido, foi gerado um novo *dataset* "`sqli.csv`" com as requisições efetuadas pela ferramenta `jSQL`, onde espera-se o *loop* reativo possa ter uma melhor performance com outras ferramentas de teste de penetração.

5.7.4 Resultados

Sintetizando os resultados das três etapas, pode-se dizer que o processo de geração de *datasets* por meio de ferramentas automatizadas, como ocorreu na **Etapa 1** (Seção 5.7.1) por meio do módulo `AppControlPanel`, mostrou-se funcional e eficiente diante dos dados gerados. Portanto, essa solução pode ser útil por diminuir a complexidade e o custo em rotular cada uma das instâncias, se comparado a um processo manual. Em paralelo, no que diz respeito a execução dos processos de preparação de dados e avaliação de algoritmos de treinamento em uma aplicação externa à solução de autoproteção (como ocorreu com *workbench* Weka) mostrou-se uma estratégia válida e viável também, pois as atividades conduzidas nesse processo podem variar entre diferentes versões de um mesmo *datasets*. Deste modo, pode-se dizer que os responsáveis por esses processos podem realizar as tarefas supracitadas de modo mais próximo ao habitual (ou seja, suas atividades cotidiana/profissionais).

Conforme pode-se observar na **Etapa 2** (Seção 5.7.2), a utilização de *datasets* públicos como ponto de partida para criação de modelos inteligentes pode ser uma solução factível por conter volume significativo de instâncias. Entretanto, pode ser necessário que dados gerados a partir da própria aplicação sejam incorporados ao *dataset* de *features*, mesmo que contenham poucas instâncias, para que possam ajustar possíveis perturbações nos modelos inteligentes. Além disso, os testes apresentados sugerem que o processo de criação de modelos inteligentes devem estar integrados e sincronizados aos processos de desenvolvimento de software, sendo

[11] `jSQL Injection`. Disponível em: https://github.com/ron190/jsql-injection, Acessado em: 4 de março de 2022

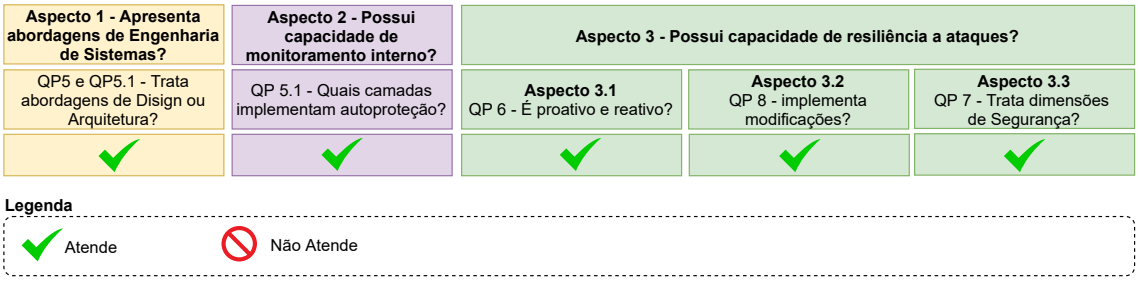
que novas atualizações da aplicação podem ser incompatíveis com os modelos de classificação criados a partir de versões anteriores da aplicação.

Por fim, os testes realizados na **Etapa 3** (Seção 5.7.3), confirmam as evidências apresentadas nos testes da **Etapa 1**, onde ferramentas de teste de penetração com diferentes características devam ser utilizadas para produzir um *dataset* com um grande volume de instâncias, aumentando assim a capacidade de generalização dos modelos de classificação. Entretanto, utilizar diferentes *loops* MAPE para detectar os mesmos tipos de ataques, como ocorreu as abordagens proativa e reativa da abordagem proposta por este trabalho, pode ser uma boa estratégia de defesa em profundidade até que se tenha um *dataset* robusto com boa capacidade de generalização.

5.8 Comparação entre a abordagem de autoproteção e trabalhos disponíveis na literatura

No que diz respeito aos aspectos avaliados nas soluções de autoproteção (veja Figura 25) que foram extraídos do mapeamento sistemático da literatura (veja Capítulo 3), é possível afirmar que a abordagem de autoproteção proposta neste trabalho atende na íntegra aos três aspectos. A Figura 57 ilustra os aspectos atendidos da abordagem, sendo que as questões de pesquisa (QPs) apresentadas nessa figura estão disponíveis na íntegra Seção 3.2.

Figura 57 – Aspectos atendidos pela abordagem proposta neste trabalho



Fonte: Autoria própria

A respeito do **Aspecto 1**, a solução de autoproteção proposta neste trabalho apresenta claras iniciativas de engenharia. Nesse contexto, a solução foi organizada em módulos (veja Figura 27), onde seus componentes internos são dotados de uma estrutura orientada a objetos. Nesse direção, pode-se dizer que os mecanismos de defesa, que operam em tempo de execução, são dotados por uma topologia organizada em *loops* MAPE-K (veja Figura 36), permitindo um modo de monitoramento não intrusivo.

Em paralelo, o **Aspecto 2** é atendido na proposta apresentada neste trabalho por meio de pontos de monitoramento, viabilizando o sensoriamento e a coleta de informações por meio de componentes internos de uma SApp. Nesse sentido, a Seção 4.1.1 elenca as tarefas que devem ser

conduzidas durante a Atividade 2, que ocorre na fase de *design* de uma SApp, para que se possa implantar tais pontos de monitoramento. Corroborando com essa afirmação, a Seção 5.2 deste capítulo exemplifica esta atividade por meio do estudo de caso conduzido para a implantação da aplicação `AppPetShop`.

Por fim, o **Aspecto 3** trata da capacidade da solução de autoproteção suportar resiliência em situações de ataques. Nesse sentido, este aspecto foi dividido em outros três aspectos especializados, a saber: (i) **Aspecto 3.1**, que trata da capacidade da solução de autoproteção empregar meios de defesa proativo e reativo. Conforme reportado na Seção 4.1.3, a abordagem apresentada neste trabalho é composta por dois *loops* MAPE, sendo um responsável pela defesa proativa e o outro pela reativa. De acordo com o relato exibido na Seção 5.7, essa estratégia mostrou-se uma solução viável como um meio de defesa em profundidade; (ii) **Aspecto 3.2**, trata da capacidade da solução de autoproteção aplicar mudanças no comportamento ou estrutura dos componentes de uma SApp como meio de contramedida em ações maliciosas. Nesta direção, embora o estudo de caso conduzido neste capítulo não tenha contemplado essa implementação em decorrência do limite de tempo (ou seja, prazo para conclusão deste mestrado acadêmico), a topologia dos *loops* MAPE empregada na abordagem proposta se mostrou capaz de atender a este aspecto; e, por fim (iii) **Aspecto 3.3**, está relacionado às dimensões de segurança que a solução de autoproteção atua, onde devem ser claramente especificadas. Neste sentido, o *design* adotado neste trabalho permite generalizar sua capacidade de autoproteção em diversas dimensões de segurança. Entretanto, como pode ser visto no estudo de caso apresentado neste capítulo, a abordagem de autoproteção foi configurada para atender aos princípios da confidencialidade e integridade da aplicação `AppPetShop`, onde os *loops* MAPE foram direcionados, por exemplo, para ataques de injeção de SQL.

5.9 Considerações finais

Neste capítulo foi apresentado um estudo de caso, que teve como objetivo avaliar a viabilidade da abordagem de autoproteção apresentada no Capítulo 4. Na Seção 5.1 foi apresentado um visão geral das tecnologias empregadas nos componentes do estudo de caso. Em seguida, na Seção 5.2 foram apresentados detalhes sobre a implementação e configuração dos pontos de monitoramento da aplicação `AppPetShop`. Na sequência, na Seção 5.3 foram apresentadas as tarefas realizadas no processo de criação e exportação dos *datasets* de *features*. Na Seção 5.4 foi abordado o processo de incorporação dos modelos de classificação aos *loops* MAPE, que compõem a abordagem de autoproteção apresentada neste trabalho. A Seção 5.5 detalhou o funcionamento do módulo `AppMape`, que é responsável por executar os *loops* MAPE. Na Seção 5.6 foram apresentadas as interfaces `View Request Log` e `View Request Behavior` do módulo `AppControlPanel`, que viabilizam o acompanhamento da evolução e performance dos módulo

`AppMape` em tempo real. A Seção 5.8 apresenta uma comparação da abordagem proposta neste trabalho com os trabalhos extraídos do mapeamento sistema da literatura elencados no Capítulo 3. Por fim, na Seção 5.7 são apresentados os testes realizados no estudo de caso e os respectivos resultados. A seguir, no Capítulo 6 são reportadas as principais contribuições desta dissertação, assim como suas perspectivas de trabalhos futuros.

6 CONCLUSÃO

Este trabalho apresentou uma abordagem de autoproteção para a camada de aplicação em SApps. Em linhas gerais, a abordagem é apoiada em processos e boas práticas que devem ser seguidos durante a fase de desenvolvimento de uma SApp, onde por meio de um *design* escalável é possível atender as atuais e futuras demandas de segurança causadas por ameaças/vulnerabilidades. Nesse sentido, a abordagem de autoproteção apresentada nesta dissertação é formada por soluções automatizadas para elaboração de modelos inteligentes de classificação que são inseridos nos *loops* de controle MAPE-K podem identificar ações maliciosas em tempo de execução e aplicar contramedidas de maneira autônoma. Além disso, o desenvolvimento de SApps pode ser considerado desafiante por englobar várias áreas de conhecimento, como ciência de dados, cibersegurança voltado a aplicações, sistemas autoadaptativos e engenharia de software. Inicialmente, o Capítulo 2 apresentou o arcabouço teórico relacionado ao tema de pesquisa, onde foram abordados os seguintes temas: na Seção 2.1 foram apresentados conceitos e definições sobre sistemas autoadaptativos, na Seção 2.2 são elencados conceitos sobre autoproteção de sistemas, e na Seção 2.3 foram abordados conceitos sobre inteligência computacional. Em seguida, no Capítulo 3 são reportados os estudos secundários disponíveis na literatura que tratam sobre autoproteção para SApps e, na sequência, o mapeamento sistemático da literatura a respeito do tema desta dissertação de mestrado. A abordagem de autoproteção proposta por este trabalho foi apresentada no Capítulo 4, sendo descrita uma visão geral da abordagem e as boas práticas que podem ser seguidas durante a implementação e a implantação da aplicação protetora. Na sequência, o Capítulo 5 apresenta um estudo de caso, cujo objetivo foi avaliar a viabilidade da abordagem proposta. Por fim, este capítulo apresenta as conclusões deste trabalho, sendo o mesmo organizado da seguinte maneira: na Seção 6.1 são apresentadas as contribuições dessa dissertação; na Seção 6.2 são reportadas as principais dificuldades enfrentadas durante a elaboração da abordagem, assim como as principais limitações; e na Seção 6.3 são elencadas as principais perspectivas futuras e oportunidades de pesquisa que merecem ser investigadas.

6.1 Contribuições da dissertação

A abordagem de autoproteção apresentada no Capítulo 4 pode ser considerada a principal contribuição deste trabalho de mestrado, pois elenca boas práticas de projeto e codificação de software. Além disso, essa abordagem também fornece processos que devem ser seguidos durante a etapa de *design* de uma SApp e executados em tempo de execução dessa aplicação. Deste modo, pode-se dizer que a abordagem proposta neste trabalho cobre algumas lacunas identificadas durante o mapeamento sistemático conduzido no Capítulo 3.

Outra contribuição relevante é o *design* escalável e soluções de ferramentas que permitem automatizar os processos presentes na abordagem apresentada no Capítulo 4. Neste sentido, o *design* orientado a objetos pode ser replicado para outros cenários e plataformas de software. Em paralelo, as soluções automatizadas apresentadas podem ser evoluídas em trabalhos futuros ao ponto de se tornarem ferramentas estáveis e funcionais.

O cenário de testes apresentado no Capítulo 5 pode ser considerado uma contribuição útil para mercado e academia, pois fornecem indicativos sobre a aplicabilidade e comportamento da abordagem em um cenário real de computação em nuvem. Entretanto, sistemas de grande escala precisam ser testados em ambientes similares de computação para avaliar sua eficácia e eficiência.

Por fim, o Capítulo 3 apresentou um mapeamento sistemático da literatura a respeito do tema de pesquisa desta dissertação, que por meio de um protocolo formal de pesquisa (veja o Apêndice A), buscou-se responder a nove questões de pesquisa. O resultado apresentado permitiu identificar as principais lacunas em aberto sobre o tema de pesquisa deste trabalho. Portanto, esse mapeamento oferece aos profissionais da indústria e comunidade científica um guia que pode nortear o desenvolvimento de novas pesquisas a respeito do tema explorado nesta dissertação.

6.2 Dificuldades e limitações

Esta seção tem por objetivo apresentar as dificuldades e limitações durante a condução deste trabalho. Basicamente, esses assuntos podem ser organizados em quatro tópicos, a saber:

Abordagem. Embora o *design* da proposta de autoproteção deste trabalho tem por objetivo permitir que os *loops* proativos e reativos identifiquem algumas variações de injeção de código, os testes apresentados na Seção 5.7 foram conduzidos por completo em torno de um tipo de ameaça (injeção de SQL). Nesse sentido, vale ressaltar que testes com outros tipos de injeção de código não foram conduzidos especificamente em virtude das limitações de tempo inerentes em um projeto de mestrado acadêmico.

Linguagem de programação. O estudo de caso apresentado no Capítulo 5 teve como principal objetivo validar a viabilidade da abordagem proposta. Entretanto, a SApp desenvolvida utilizou como base a linguagem de programação Java e o *framework* `SpringBoot`. Portanto, pode-se dizer que é necessário testar os processos, *design* e ferramentas automatizadas que fazem parte da abordagem proposta neste trabalho dentro de outras plataformas e linguagens de programação para validar sua replicabilidade.

Datasets. Uma dificuldade identificada durante a condução do estudo de caso apresentado no Capítulo 5 está relacionada a ausência de *datasets* que contenham comportamentos

relacionados aos componentes internos de uma SApp. Os *datasets* encontrados na literatura estão limitados ao tráfego de rede ou requisições HTTP. Portanto, foi um desafio criar o *dataset* responsável pelo treinamento do modelo de classificação para o *loop* reativo. Deste modo, outras ferramentas de testes de penetração merecem ser investigadas para serem utilizadas na formação de um *dataset* mais robusto.

Classificação de novas anomalias. Os modelos de treinamento apresentados neste trabalho são baseados em aprendizado supervisionado e possuem grande potencial de generalização para as categorias ameaças/vulnerabilidades conhecidas na literatura/mercado. Entretanto, em situações que existam ameaças/vulnerabilidades completamente desconhecidas (ou seja, dias de zero) e que não resultem em indisponibilidades ou anomalias perceptíveis aos administradores de sistema, pode-se levar um tempo elevado para haver intervenções significativas. Neste contexto, a abordagem de autoproteção apresentada nesta dissertação pode falhar em classificar anomalias completamente desconhecidas.

6.3 Trabalhos futuros

A fim de validar a replicabilidade da abordagem apresentada neste trabalho sugere-se que novos estudos de caso sejam conduzidos, onde pretende-se utilizar as soluções desenvolvidas junto com SApps implementados em outras plataformas e linguagens de programação. Nesse sentido, o módulo `LibSelfProtection` (veja Figura 27) deverá ser reescrito para garantir a compatibilidade da abordagem com o cenário nativo da SApp desenvolvida. Em paralelo, aplicações baseadas em serviços e computação móvel devem ser implementadas e incluídas nos estudos de caso para avaliar a generalização do *design* elencado na abordagem deste trabalho de mestrado.

No estudo de caso apresentado no Capítulo 5, os processos relacionados a fase de *design* foram conduzidos durante as fases de concepção, projeto e codificação da aplicação `AppPetShop`. No entanto, os resultados dos testes apresentados na Seção 5.7 evidenciam que os processos de ciência de dados devem estar presentes na fase de manutenção do software, onde podem ser lançadas *releases* recorrentemente. Nessa direção, pretende-se investigar em trabalhos futuros a condução da abordagem durante outras fases do desenvolvimento de software, assim como em diferentes metodologias de desenvolvimento de software.

A respeito da aplicação `AppPetShop`, durante a fase de *design* foi identificada a vulnerabilidade de injeção de SQL. Entretanto, os *loops* MAPE-K estão aptos a detectar outros tipos de vulnerabilidades como XPath, SSI, *Formart String*, LDAPi, CRLFi e *Buffer Overflow*. Essas vulnerabilidades não foram tratadas neste trabalho de mestrado por restrições de tempo (ou seja, prazo para conclusão deste trabalho). Assim, pretende-se conduzir novas investigações e

estudos de caso que permitam avaliar o comportamento da abordagem de autoproteção proposta diante de diferentes ameaças/vulnerabilidades. Em paralelo, pretende-se também utilizar outras ferramentas automatizadas de testes de penetração para auxílio nas fases de *design* e *runtime* da abordagem.

Em relação aos processos de ciência de dados que estão presentes na fase de *design* da abordagem, pretende-se investigar outras técnicas de aprendizagem como não supervisionada, semi-supervisionada e por reforço. Onde o objetivo será incluir à abordagem a capacidade de detectar anomalias que sejam completamente novas. Em outras palavras, o objetivo é permitir que os administradores de sistemas sejam alertados mesmo que o *dataset* de treinamento não tenham instâncias suficientes para generalizar ataques de dia zero.

Por fim, pretende-se acompanhar também a evolução da área de pesquisa deste projeto através da atualização do mapeamento sistemático reportado no Capítulo 3.

REFERÊNCIAS

AFTERGOOD, S. Cybersecurity: The cold war online. *Nature*, v. 547, p. 30–31, jul. 2017. Disponível em: https://doi.org/10.1038/547030a. Citado na página 34.

ALLEN, D. M.; GOLOUBEW, D. Customer self-remediation of proactive network issue detection and notification. *In*: DEGEN, H.; REINERMAN-JONES, L. (Ed.). *Artificial Intelligence in HCI*. Cham: Springer International Publishing, 2020. p. 197–210. ISBN 978-3-030-50334-5. Disponível em: https://doi.org/10.1007/978-3-030-50334-5_13. Citado na página 13.

ALPAYDIN, E. *Introduction to Machine Learning*. [*s.l.*]: The MIT Press, 2014. ISBN 0262028182. Citado 2 vezes na(s) página(s) 31 e 34.

ARIAS-CABARCOS, P.; KRUPITZER, C.; BECKER, C. A survey on adaptive authentication. *ACM Computing Surveys*, Association for Computing Machinery, New York, NY, United States, v. 52, n. 4, p. 1–30, set. 2019. ISSN 0360-0300. Disponível em: https://doi.org/10.1145/3336117. Citado na página 41.

BASS, L.; KAZMAN, R.; CLEMENTS, P. *Software Architecture in Practice*. [*s.l.*]: Addison Wesley, 2012. ISBN 0321815734. Citado na página 62.

Benzaïd, C.; Boukhalfa, M.; Taleb, T. Robust self-protection against application-layer (d)dos attacks in sdn environment. *In*: *2020 IEEE Wireless Communications and Networking Conference (WCNC)*. [*s.n.*], 2020. p. 1–6. ISSN 1558-2612. Disponível em: https://doi.org/10.1109/WCNC45663.2020.9120472. Citado 2 vezes na(s) página(s) 46 e 59.

BENZAID, C.; TALEB, T. Ai for beyond 5g networks: A cyber-security defense or offense enabler? *IEEE Network*, v. 34, n. 6, p. 140–147, nov. 2020. ISSN 1558-156X. Disponível em: https://doi.org/10.1109/MNET.011.2000088. Citado na página 13.

BHAGWANI, H. *et al.* Automated classification of web-application attacks for intrusion detection. *In*: *Security, Privacy, and Applied Cryptography Engineering*. Cham: Springer International Publishing, 2019. p. 123–141. ISBN 978-3-030-35869-3. Disponível em: https://doi.org/10.1007/978-3-030-35869-3_10. Citado 3 vezes na(s) página(s) 37, 38 e 87.

BJöRCK, F. *et al.* Cyber resilience – fundamentals for a definition. *In*: *New Contributions in Information Systems and Technologies*. Cham: Springer International Publishing, 2015. p. 311–316. ISBN 978-3-319-16486-1. Disponível em: https://doi.org/10.1007/978-3-319-16486-1_31. Citado na página 70.

BOUDKO, S.; ABIE, H. Adaptive cybersecurity framework for healthcare internet of things. *In*: *2019 13th International Symposium on Medical Information and Communication Technology (ISMICT)*. [*s.n.*], 2019. p. 1–6. ISBN 978-1-7281-2343-1. ISSN 2326-8301. Disponível em: https://doi.org/10.1109/ISMICT.2019.8743905. Citado 2 vezes na(s) página(s) 11 e 13.

BRAGA, T. R. M. *et al. Redes Autonomicas*. 2006. Disponível em: https://www.ppgee.ufmg.br/documentos/PublicacoesDefesas/707/minicurso_sbrc2006.pdf, Acesso em: 03 fev. 2022. Citado na página 16.

BROWNLEE, J. *An Introduction to Feature Selection*. 2021. Disponível em: https://machinelearningmastery.com/an-introduction-to-feature-selection, Acesso em: 03 fev. 2022. Citado na página 147.

BUCZAK, A. L.; GUVEN, E. A survey of data mining and machine learning methods for cyber security intrusion detection. *IEEE Communications Surveys & Tutorials*, Institute of Electrical and Electronics Engineers (IEEE), v. 18, n. 2, p. 1153–1176, 2016. ISSN 1553-877X. Disponível em: https://doi.org/10.1109/COMST.2015.2494502. Citado na página 59.

CALVERT, C. L.; KHOSHGOFTAAR, T. M. Impact of class distribution on the detection of slow http dos attacks using big data. *Journal of Big Data*, v. 6, n. 1, p. 67, Julho 2019. ISSN 2196-1115. Disponível em: https://doi.org/10.1186/s40537-019-0230-3. Citado 3 vezes na(s) página(s) 11, 13 e 46.

CHENG, B. H. C. *et al.* Software engineering for self-adaptive systems: A research roadmap. *In*: ____. *Software Engineering for Self-Adaptive Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009. p. 1–26. ISBN 978-3-642-02161-9. Disponível em: https://doi.org/10.1007/978-3-642-02161-9_1. Citado na página 20.

CISAR, P.; CISAR, S. M. The framework of runtime application self-protection technology. *In*: *2016 IEEE 17th International Symposium on Computational Intelligence and Informatics (CINTI)*. IEEE, 2016. p. 81–86. ISBN 978-1-5090-3909-8. ISSN 2471-9269. Disponível em: https://doi.org/10.1109/CINTI.2016.7846383. Citado na página 62.

CLEMENT, S. J.; MCKEE, D. W.; XU, J. Service-oriented reference architecture for smart cities. *In*: *2017 IEEE Symposium on Service-Oriented System Engineering (SOSE)*. IEEE, 2017. ISBN 978-1-5090-6321-5. Disponível em: https://doi.org/10.1109/SOSE.2017.29. Citado na página 11.

CROSS, M. *Developer's Guide to Web Application Security*. [*s.l.*]: Syngress Publishing, Inc, 2007. ISBN 9781597490610. Citado na página 25.

CWE. *2019 CWE Top 25 Most Dangerous Software Errors*. 2019. Disponível em: https://cwe.mitre.org/top25/archive/2019/2019_cwe_top25.html, Acesso em: 03 fev. 2022. Citado na página 25.

DENG, L. *et al.* RETRACTED ARTICLE: mobile network intrusion detection for iot system based on transfer learning algorithm. *Cluster Computing*, v. 22, n. 4, p. 9889, 2019. ISSN 1573-7543. Disponível em: https://doi.org/10.1007/s10586-018-1847-2. Citado na página 13.

DIESTE, O.; GRIMáN, A.; JURISTO, N. Developing search strategies for detecting relevant experiments. *Empirical Software Engineering*, Kluwer Academic Publishers, Hingham, MA, USA, v. 14, n. 5, p. 513–539, 2009. ISSN 1382-3256. Disponível em: https://doi.org/10.1007/s10664-008-9091-7. Citado na página 135.

DYBA, T.; DINGSOYR, T.; HANSSEN, G. K. Applying systematic reviews to diverse study types: An experience report. *In*: *Proceedings of the 1st Empirical Software Engineering and Measurement (ESEM'07)*. Madri, Spain: IEEE Computer Society, 2007. p. 225–234. ISBN 978-0-7695-2886-1. ISSN 0-7695-2886-4. Disponível em: https://doi.org/10.1109/ESEM.2007.59. Citado na página 45.

ELEUTéRIO, J. D. A. S. *Técnicas de sistemas autônomos e autoadaptativos para apoiar linhas de produtos de software dinâmicas*. Tese (Doutorado) — Instituto de Ciencias Matemáticas e de Computação - Universidade de São Paulo, 2017. Disponível em: http://bdtd.ibict.br/vufind/Record/CAMP_87988daa5bce050766fc26c70c2b093e, Acesso em: 03 fev. 2022. Citado na página 84.

ELKHODARY, A.; WHITTLE, J. A survey of approaches to adaptive application security. *In*: *International Workshop on Software Engineering for Adaptive and Self-Managing Systems*. IEEE, 2007. ISSN 2157-2321. Disponível em: https://doi.org/10.1109/SEAMS.2007.2. Citado 2 vezes na(s) página(s) 28 e 29.

FAYAD, M. *Implementing application frameworks : object-oriented frameworks at work*. New York: Wiley, 1999. ISBN 0471252018. Citado na página 50.

FERBER, S.; HAAG, J.; SAVOLAINEN, J. Feature interaction and dependencies: Modeling features for reengineering a legacy product line. *In*: CHASTEK, G. J. (Ed.). *Software Product Lines*. Springer Berlin Heidelberg, 2002. v. 2379, p. 235–256. ISBN 978-3-540-45652-0. Disponível em: https://doi.org/10.1007/3-540-45652-X_15. Citado na página 84.

FIORE, S. *et al.* An integrated big and fast data analytics platform for smart urban transportation management. *IEEE Access*, Institute of Electrical and Electronics Engineers (IEEE), v. 7, p. 117652–117677, ago. 2019. ISSN 2169-3536. Disponível em: https://doi.org/10.1109/ACCESS.2019.2936941. Citado na página 11.

FOROUGHI, F.; LUKSCH, P. Data science methodology for cybersecurity projects. p. 01–14, 02 2018. Disponível em: http://dx.doi.org/10.5121/csit.2018.80401. Citado 2 vezes na(s) página(s) 34 e 35.

FOWLER, M. *Software Architecture Guide*. 2019. Disponível em: https://martinfowler.com/architecture, Acesso em: 03 fev. 2022. Citado na página 50.

GREGG, D. G.; KULKARNI, U. R.; VINZÉ, A. S. Understanding the philosophical underpinnings of software engineering research in information systems. *Information Systems Frontiers*, Springer Science and Business Media LLC, v. 3, n. 2, p. 169–183, jun. 2001. Disponível em: https://doi.org/10.1023/A:1011491322406. Citado na página 56.

GéRON, A. *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. [*s.l.*]: O' Reilly, 2007. Citado 4 vezes na(s) página(s) 31, 32, 33 e 34.

HUEBSCHER, M. C.; MCCANN, J. A. A survey of autonomic computing—degrees, models, and applications. *ACM Computing Surveys*, Association for Computing Machinery (ACM), v. 40, n. 3, p. 1–28, ago. 2008. Disponível em: https://doi.org/10.1145/1380584.1380585. Citado na página 12.

IBM. *An architectural blueprint for autonomic computing*. 2005. Disponível em: http://www.ginkgo-networks.com/IMG/pdf/AC_Blueprint_White_Paper_V7.pdf, Acesso em: 03 fev. 2022. Citado 4 vezes na(s) página(s) 15, 19, 20 e 30.

IBM. *Application Programming Interface (API)*. 2020. Disponível em: https://www.ibm.com/cloud/learn/api, Acesso em: 03 fev. 2022. Citado na página 61.

KACI, A.; RACHEDI, A. Mc-track: A cloud based data oriented vehicular tracking system with adaptive security. *In*: *2019 IEEE Global Communications Conference (GLOBECOM)*. [*s.n.*], 2019. p. 1–6. ISBN 978-1-7281-0962-6. ISSN 2576-6813. Disponível em: https://doi.org/10.1109/GLOBECOM38437.2019.9013977. Citado 3 vezes na(s) página(s) 11, 12 e 13.

KEPHART, J.; CHESS, D. The vision of autonomic computing. *Computer*, Institute of Electrical and Electronics Engineers (IEEE), v. 36, n. 1, p. 41–50, jan. 2003. ISSN 1558-0814. Disponível em: https://doi.org/10.1109/MC.2003.1160055. Citado 4 vezes na(s) página(s) 15, 17, 19 e 20.

KERN, C.; KESAVAN, A.; DASWANI, N. *Foundations of Security*. [*s.l.*]: Apress, 2007. ISBN 1590597842. Citado 2 vezes na(s) página(s) 12 e 24.

KHALID, A. *et al.* Survey of frameworks, architectures and techniques in autonomic computing. *In*: *2009 Fifth International Conference on Autonomic and Autonomous Systems*. IEEE, 2009. p. 220–225. ISBN 978-0-7695-3584-5. ISSN 2168-1864. Disponível em: https://doi.org/10.1109/ICAS.2009.38. Citado 2 vezes na(s) página(s) 15 e 17.

KHANJI, S.; KHATTAK, A. Towards a novel intrusion detection architecture using artificial intelligence. *In*: *Proceedings of the 2020 9th International Conference on Software and Information Engineering (ICSIE)*. New York, NY, USA: Association for Computing Machinery, 2020. (ICSIE 2020), p. 185–189. ISBN 9781450377218. Disponível em: https://doi.org/10.1145/3436829.3436842. Citado na página 13.

KITCHENHAM, B.; CHARTERS, S. *Guidelines for performing Systematic Literature Reviews in Software Engineering*. [*s.l.*], 2007. Disponível em: https://www.elsevier.com/__data/promis_misc/525444systematicreviewsguide.pdf, Acesso em: 03 fev. 2022. Citado 3 vezes na(s) página(s) 39, 45 e 135.

KITCHENHAM, B. *et al.* Systematic literature reviews in software engineering – a tertiary study. *Information and Software Technology*, v. 52, n. 8, p. 792–805, Aggosto 2010. ISSN 0950-5849. Disponível em: https://doi.org/10.1016/j.infsof.2010.03.006. Citado na página 39.

KITCHENHAM, B. A.; DYBA, T.; JORGENSEN, M. Evidence-based software engineering. IEEE Computer Society, USA, p. 273–281, msio 2004. ISSN 0270-5257. Disponível em: https://doi.org/10.1109/ICSE.2004.1317449. Citado na página 12.

KRUPITZER, C. *et al.* A survey on engineering approaches for self-adaptive systems. *Pervasive and Mobile Computing*, Elsevier BV, v. 17, p. 184–206, feb 2015. ISSN 1574-1192. Disponível em: https://doi.org/10.1016/j.pmcj.2014.09.009. Citado 2 vezes na(s) página(s) 16 e 22.

LA, H. J.; KIM, S. D. A machine learning framework for adaptive fintech security provisioning. *Journal of Internet Technology*, v. 19, n. 5, p. 1545–1553, set. 2018. ISSN 1607-9264. Disponível em: https://jit.ndhu.edu.tw/article/view/1775, Acesso em: 03 fev. 2022. Citado 2 vezes na(s) página(s) 11 e 13.

LAKATOS, E. *Fundamentos de metodologia cientifica*. Sao Paulo: Atlas, 2003. ISBN 8522433976. Citado na página 49.

LEMOS, R. de *et al.* Software engineering for self-adaptive systems: A second research roadmap. *In*: *Software Engineering for Self-Adaptive Systems II*. Springer Berlin Heidelberg, 2013. v. 7475, p. 1–32. ISBN 978-3-642-35813-5. Disponível em: https://doi.org/10.1007/978-3-642-35813-5_1. Citado 2 vezes na(s) página(s) 20 e 21.

MA, X. *et al.* An API semantics-aware malware detection method based on deep learning. *Security and Communication Networks*, Hindawi Limited, v. 2019, p. 1–9, nov. 2019. ISSN 1939-0114. Disponível em: https://doi.org/10.1155/2019/1315047. Citado na página 46.

MALEK, S. *et al.* Security and software engineering. *In*: *Handbook of Software Engineering*. Springer International Publishing, 2019. p. 445–489. ISBN 978-3-030-00262-6. Disponível em: https://doi.org/10.1007/978-3-030-00262-6_12. Citado 2 vezes na(s) página(s) 26 e 27.

MARTINS, R. *et al.* A self-protecting approach for service-oriented mobile applications. *In*: INSTICC. *Proceedings of the 23rd International Conference on Enterprise Information Systems - Volume 2: ICEIS,*. SCITEPRESS - Science and Technology Publications, 2021. p. 313–320. ISBN 978-989-758-509-8. ISSN 2184-4992. Disponível em: https://doi.org/10.5220/0010448603130320. Citado na página 87.

MISHRA, A. K.; KUMAR, A. Performance-based comparative analysis of open source vulnerability testing tools for web database applications. *In*: *2020 11th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*. IEEE, 2020. ISBN 978-1-7281-6851-7. Disponível em: https://doi.org/10.1109/icccnt49239.2020.9225324. Citado na página 117.

MOHRI, M. N. Y. U. *et al. Foundations of Machine Learning*. [*s.l.*]: MIT Press Ltd, 2012. ISBN 026201825X. Citado 5 vezes na(s) página(s) 31, 32, 33, 34 e 69.

MONGODB. *Security Checklist*. 2021. Disponível em: https://owasp.org/www-project-mobile-top-10, Acesso em: 03 fev. 2022. Citado na página 97.

MONTRIEUX, L.; LEMOS, R. de; BAILEY, C. Challenges in engineering self-adaptive authorisation infrastructures. *In*: *Engineering Adaptive Software Systems*. Springer Singapore, 2019. p. 57–94. ISBN 978-981-13-2185-6. Disponível em: https://doi.org/10.1007/978-981-13-2185-6_3. Citado na página 12.

NORD, J. H.; KOOHANG, A.; PALISZKIEWICZ, J. The internet of things: Review and theoretical framework. *Expert Systems with Applications*, Elsevier BV, v. 133, p. 97–108, nov. 2019. ISSN 0957-4174. Disponível em: http://doi.org/10.1016/j.eswa.2019.05.014. Citado na página 11.

NORVIG, P.; RUSSELL, S. *Artificial Intelligence: A Modern Approach, Global Edition*. [*s.l.*]: Pearson, 2021. ISBN 1292401133. Citado 5 vezes na(s) página(s) 12, 31, 33, 34 e 59.

OWASP. *Code Review Guide 2.0: Release*. 2017. Disponível em: https://owasp.org/www-pdf-archive/OWASP_Code_Review_Guide_v2.pdf, Acesso em: 03 fev. 2022. Citado na página 26.

OWASP. *OWASP Top 10 - 2017: The Ten Most Critical Web Application Security Risks*. 2017. Disponível em: https://wiki.owasp.org/images/0/06/OWASP_Top_10-2017-pt_pt.pdf, Acesso em: 03 fev. 2022. Citado 2 vezes na(s) página(s) 26 e 101.

OWASP. *Injeção SQL*. 2021. Disponível em: https://owasp.org/www-community/attacks/SQL_Injection, Acesso em: 03 fev. 2022. Citado na página 102.

OWASP. *OWASP Application Security Verification Standard*. 2021. Disponível em: https://owasp.org/www-project-application-security-verification-standard, Acesso em: 03 fev. 2022. Citado na página 80.

OWASP. *OWASP Mobile Top 10 2016*. 2021. Disponível em: https://docs.mongodb.com/manual/administration/security-checklist, Acesso em: 03 fev. 2022. Citado na página 26.

OWASP. *OWASP Proactive Controls*. 2021. Disponível em: https://owasp.org/www-project-proactive-controls, Acesso em: 03 fev. 2022. Citado 3 vezes na(s) página(s) 13, 79 e 80.

OWASP. *OWASP Risk Rating Methodology*. 2021. Disponível em: https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology, Acesso em: 03 fev. 2022,. Citado na página 78.

OWASP. *OWASP Top Ten*. 2022. Disponível em: https://owasp.org/www-project-top-ten, Acesso em: 03 fev. 2022. Citado na página 26.

PASSINI, W. F. *et al.* Design of frameworks for self-adaptive service-oriented applications: A systematic analysis. *Software: Practice and Experience*, Wiley, v. 52, p. 1–315, jan. 2021. Disponível em: https://doi.org/10.1002/spe.3013. Citado na página 55.

PERRY, D.; SIM, S.; EASTERBROOK, S. Case studies for software engineers. *In*: *Proceedings. 26th International Conference on Software Engineering*. IEEE Comput. Soc, 2004. ISBN 0-7695-2163-0. ISSN 0270-5257. Disponível em: https://doi.org/10.1109/icse.2004.1317512. Citado na página 56.

PETERSEN, K. *et al.* Systematic mapping studies in software engineering. *In*: *12th International Conference on Evaluation and Assessment in Software Engineering*. Swindon, UK: BCS Learning & Development Ltd., 2008. p. 68–77. Disponível em: https://www.scienceopen.com/hosted-document?doi=10.14236/ewic/EASE2008.8, Acesso em: 03 fev. 2022. Citado 5 vezes na(s) página(s) 12, 39, 43, 44 e 133.

PETERSEN, K.; VAKKALANKA, S.; KUZNIARZ, L. Guidelines for conducting systematic mapping studies in software engineering: An update. *Information and Software Technology*, v. 64, p. 1–18, ago. 2015. ISSN 0950-5849. Disponível em: https://doi.org/10.1016/j.infsof.2015.03.007. Citado 8 vezes na(s) página(s) 12, 39, 43, 44, 45, 133, 135 e 137.

REDDY, Y. C. A. P.; VISWANATH, P.; REDDY, B. E. Semi-supervised learning: a brief review. *International Journal of Engineering & Technology*, Science Publishing Corporation, v. 7, n. 1.8, p. 81–85, fev. 2018. ISSN 2227-524X. Disponível em: https://doi.org/10.14419/ijet.v7i1.8.9977. Citado na página 33.

RESTUCCIA, F. *et al.* The role of machine learning and radio reconfigurability in the quest for wireless security. *In*: ____. *Proactive and Dynamic Network Defense*. Cham: Springer International Publishing, 2019. p. 191–221. ISBN 978-3-030-10597-6. Disponível em: https://doi.org/10.1007/978-3-030-10597-6_8. Citado na página 13.

SAHATQIJA, K. *et al.* Comparison between relational and NOSQL databases. *In*: *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. IEEE, 2018. p. 216–221. ISBN 978-953-233-095-3. Disponível em: https://doi.org/10.23919/mipro.2018.8400041. Citado 2 vezes na(s) página(s) 96 e 97.

SALEHIE, M.; TAHVILDARI, L. Self-adaptive software: Landscape and research challenges. *ACM Transactions on Autonomous and Adaptive Systems*, ACM, New York, NY, USA, v. 4, n. 2, p. 14:1–14:42, maio 2009. ISSN 1556-4665. Disponível em: http://doi.acm.org/10.1145/1516533.1516538. Citado 5 vezes na(s) página(s) 15, 16, 17, 18 e 19.

SALLOUM, S. A. *et al.* Machine learning and deep learning techniques for cybersecurity: A review. *In*: *Proceedings of the International Conference on Artificial Intelligence and Computer Vision (AICV2020)*. Cham: Springer International Publishing, 2020. p. 50–57. ISBN 978-3-030-44289-7. Disponível em: https://doi.org/10.1007/978-3-030-44289-7_5. Citado na página 38.

SARICA, A.; ANGIN, P. Explainable security in sdn-based iot networks. *Sensors*, v. 20, n. 24, p. 1–30, 2020. ISSN 1424-8220. Disponível em: https://doi.org/10.3390/s20247326. Citado 2 vezes na(s) página(s) 11 e 13.

SARKER, I. H. *et al.* Cybersecurity data science: an overview from machine learning perspective. *Journal of Big Data*, Springer Science and Business Media LLC, v. 7, n. 1, jul. 2020. Disponível em: https://doi.org/10.1186/s40537-020-00318-5. Citado 3 vezes na(s) página(s) 34, 36 e 37.

SCHNEIER. *Secrets and Lies 15th Annivers*. [*s.l.*]: John Wiley & Sons, 2015. ISBN 1119092434. Citado 2 vezes na(s) página(s) 11 e 24.

SEVOCAB. *Software and Systems Engineering Vocabulary (SEVOCAB)*. 2021. Disponível em: https://pascal.computer.org/sev_display/index.action, Acesso em: 03 fev. 2022. Citado 4 vezes na(s) página(s) 49, 54, 55 e 56.

SHAH, B.; SHAH, M. A survey on machine learning and deep learning based approaches for sarcasm identification in social media. *In*: *Data Science and Intelligent Applications*. Singapore: Springer Singapore, 2020. v. 52, p. 247–259. ISBN 978-981-15-4474-3. Disponível em: https://doi.org/10.1007/978-981-15-4474-3_29. Citado na página 33.

SHALEV-SHWARTZ, S. B.-D. S. *Understanding Machine Learning*. [*s.l.*]: Cambridge University Pr., 2014. ISBN 1107057132. Citado na página 31.

SOMMERVILLE, I. *Software engineering*. Boston: Pearson, 2019. ISBN 9780133943030. Citado 2 vezes na(s) página(s) 78 e 79.

TANENBAUM, A. *Distributed systems : principles and paradigms*. Upper Saddle River, NJ: Pearson Prentice Hall, 2007. ISBN 0132392275. Citado na página 62.

Tariq, U. *et al.* Context-aware autonomous security assertion for industrial iot. *IEEE Access*, v. 8, p. 191785–191794, 2020. ISSN 2169-3536. Disponível em: https://doi.org/10.1109/ACCESS.2020.3032436. Citado na página 13.

TORRANO-GIMENEZ, C.; PEREZ-VILLEGAS, A.; ALVAREZ, G. Torpeda: Un conjunto de datos ampliable para la evaluación de cortafuegos de aplicaciones web. *XII Reunión Española sobre Criptología y Seguridad de la Información (RECSI 2012). San Sebastián*, 2012. Disponível em: https://recsi2012.mondragon.edu/es/programa/recsi2012_submission_73.pdf, Acesso em: 03 fev. 2022. Citado na página 38.

TUN, T. T. *et al.* Requirements and specifications for adaptive security: Concepts and analysis. *In*: *2018 IEEE/ACM 13th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. [*s.l.*: *s.n.*], 2018. p. 161–171. ISBN 978-1-4503-5715-9. ISSN 2157-2305. Disponível em: https://ieeexplore.ieee.org/document/8595393, Acesso em: 03 fev. 2022. Citado na página 13.

TZIAKOURIS, G.; BAHSOON, R.; BABAR, M. A. A survey on self-adaptive security for large-scale open environments. *ACM Computing Surveys*, Association for Computing Machinery, New York, NY, USA, v. 51, n. 5, Out. 2018. ISSN 0360-0300. Disponível em: https://doi.org/10.1145/3234148. Citado 2 vezes na(s) página(s) 28 e 42.

VELDHOVEN, Z. V.; VANTHIENEN, J. Designing a comprehensive understanding of digital transformation and its impact. *In*: *Humanizing Technology for a Sustainable Society*. University of Maribor Press, 2019. ISBN 978-961-286-280-0. Disponível em: https://doi.org/10.18690/978-961-286-280-0.39. Citado na página 11.

WITTEN, I. H. *et al. Data mining : practical machine learning tools and techniques*. Burlington, MA: Morgan Kaufmann, 2011. ISBN 9780123748560. Citado 4 vezes na(s) página(s) 59, 84, 85 e 146.

YANG, J.; ZHOU, M.; CUI, B. MLAB-BiLSTM: online web attack detection via attention-based deep neural networks. *In*: YU, S.; MUELLER, P.; QIAN, J. (Ed.). *Security and Privacy in Digital Economy*. Singapore: Springer Singapore, 2020. p. 482–492. ISBN 978-981-15-9129-7. Disponível em: https://doi.org/10.1007/978-981-15-9129-7_33. Citado 4 vezes na(s) página(s) 11, 12, 13 e 47.

YUAN, E.; ESFAHANI, N.; MALEK, S. A systematic survey of self-protecting software systems. *ACM Transactions on Autonomous and Adaptive Systems*, Association for Computing Machinery (ACM), v. 8, n. 4, p. 1–41, jan. 2014. Disponível em: https://doi.org/10.1145/2555611. Citado 3 vezes na(s) página(s) 29, 30 e 42.

YUAN, E.; MALEK, S. A taxonomy and survey of self-protecting software systems. *In*: *Proceedings of the 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. Zurich, Switzerland: IEEE Press, 2012. (SEAMS '12), p. 109–118. ISBN 9781467317870. Disponível em: https://doi.org/10.1109/SEAMS.2012.6224397. Citado 2 vezes na(s) página(s) 42 e 43.

ZHOU, Z. *et al.* Endogenous security defense against deductive attack: When artificial intelligence meets active defense for online service. *IEEE Communications Magazine*, v. 58, n. 6, p. 58–64, jun. 2020. ISSN 1558-1896. Disponível em: https://doi.org/10.1109/MCOM.001.1900367. Citado 2 vezes na(s) página(s) 11 e 46.

ZSEBY, T.; PFEFFER, H.; STEGLICH, S. Concepts for self-protection. *In*: ____. *Autonomic Computing and Networking*. Boston, MA: Springer US, 2009. p. 355–380. ISBN 978-0-387-89828-5. Disponível em: https://doi.org/10.1007/978-0-387-89828-5_15. Citado na página 29.

APÊNDICE A – PROTOCOLO DE PESQUISA

A.1 Introdução

Este estudo visa estabelecer uma visão ampla e justa sobre o projeto de abordagens de autoproteção para o domínio aplicações *desktop*, móveis e Web baseadas em serviços. Essas aplicações serão referenciadas deste ponto em diante como SApps. Para isso, foi adotada a técnica de SMS, sendo que para conduzi-lo, o processo proposto por (PETERSEN *et al.*, 2008; PETERSEN; VAKKALANKA; KUZNIARZ, 2015) foi utilizado, o qual é composto por cinco passos principais:

- **[Passo 1]** Definição das Questões de Pesquisa. Neste passo são definidos os objetivos da pesquisa e o protocolo de mapeamento sistemático, que representa um plano pré-determinado que descreve questões de pesquisa, assim como o mapeamento sistemático deve ser conduzido;
- **[Passo 2]** Condução da Pesquisa. Neste passo os estudos primários são recuperados e identificados;
- **[Passo 3]** Triagem dos Artigos. Neste passo os estudos identificados são avaliados de acordo com os critérios de inclusão e exclusão definidos no protocolo de pesquisa;
- **[Passo 4]** Elaboração dos resumos. Neste passo os estudos incluídos são analisados e classificados; e
- **[Passo 5]** Extração de dados e mapeamento dos estudos. Neste passo ocorre a construção da representação final do mapeamento, que deve ser realizada com base no esquema final criado no Passo 4.

Como forma de apresentar os passos supracitados, este documento foi organizado como segue. Na Seção A.2 é apresentada as QPs proposta para este SMS. Já na Seção A.3 é reportada a estratégia de pesquisa utilizada neste SMS, sendo destacada a *string* de busca, as bases de pesquisa utilizadas e os critérios de inclusão e exclusão utilizados para selecionar os estudos. Em seguida, na Seção A.4 é mostrado como os dados foram coletados e sintetizados. Na Seção A.5 são reportadas as limitações deste tipo de trabalho. Por fim, na Seção A.6 é apresentado um breve relato sobre como os resultados devem ser reportados.

A.2 Questões de pesquisa

O objetivo principal deste SMS é identificar estudos primários que relatem o projeto de abordagens de autoproteção para SApps. Este mapeamento também visa obter uma visão abrangente sobre as características desse tipo aplicação. Para isso, as seguintes Questões de Pesquisa (QP) foram elaboradas:

- **QP 1:** Quais são as soluções de autoproteção que foram elaboradas para apoiar o desenvolvimento de SApps?
  O objetivo desta questão é compreender quais são os tipos de soluções elaboradas. Por exemplo, frameworks, bibliotecas, ferramentas, plugins, arquiteturas, modelos, abordagens, plataformas de software, entre outros.

- **QP 2:** Quais são os domínios de aplicação que se beneficiaram por ter soluções de autoproteção para SApps?
  O objetivo dessa questão é identificar os domínios de aplicação que propuseram ou utilizaram soluções de autoproteção para SApps. Por exemplo: sistemas de cuidados de saúde, agência de viagens, entre outros

- **QP 3:** Quais são os atributos de qualidade utilizados no projeto de soluções de autoproteção para SApps?
  O objetivo dessa questão é entender quais são os atributos de qualidade foram utilizados no projeto de soluções de autoproteção para SApps. A partir das respostas para essa questão, as soluções podem ser analisadas ou desenvolvidas com base nos atributos adotados.

- **QP 4:** Como as soluções de autoproteção para SApps têm sido avaliadas?
  O objetivo desta questão é compreender quais técnicas foram utilizadas para avaliar tais soluções. Por exemplo, estudos de caso, provas de conceito, experimentos, entre outros.

  - **QP 4.1:** Quais são as evidências que motivaram a adoção de soluções de autoproteção em SApps?
    O objetivo desta questão é compreender o nível de maturidade de tais soluções.

- **QP 5:** Como as soluções de autoproteção para SApps têm sido projetadas?
  O objetivo dessa questão é entender qual(is) técnica(s) está(ão) sendo utilizada(s) para projetar SApps tanto em fase de projeto quanto de execução.

  - **QP 5.1:** Como as soluções de autoproteção para SApps têm sido organizadas?
    O objetivo desta questão é compreender as soluções para SApps foram organizadas do ponto de vista arquitetural.

- **QP 6:** Quais estratégias foram usadas no projeto de soluções de autoproteção para SApps?
  O objetivo desta questão é compreender como as SApps reagem frente aos tipos de ameaças e/ou ataques. Por exemplo, as SApps tem comportamento reativo, proativo, ou ambos.

- **QP 7:** Quais são as dimensões da segurança das soluções de autoproteção abordadas em SApps?
  O objetivo desta questão é compreender as dimensões de segurança aplicadas em tais soluções. Por exemplo, controle de acesso, autenticação, não repúdio, confidencialidade de dados, segurança de comunicação, integridade de dados, disponibilidade, privacidade, autorização, entre outros.

- **QP 8:** Quais são as modificações realizadas pelas soluções de autoproteção em SApps?
  O objetivo desta questão é compreender as dimensões de modificação aplicadas em tais soluções. Por exemplo, estrutural, comportamental, arquitetural, contexto, mista.

- **QP 9:** Quais são as técnicas de aprendizado abordadas nas soluções de autoproteção para SApps?
  O objetivo desta questão é compreender as técnicas de aprendizado aplicadas em tais soluções. Onde as mesmas são aplicadas e quais abordagens são utilizadas (por exemplo, supervisionada, semi-supervisionado, não-supervisionada, por reforço).

A.3 Estratégia de pesquisa

A estratégia de pesquisa foi definida com base nas QPs apresentadas na Seção A.2. Em resumo, essa estratégia é composta por critério de seleção das bases de pesquisa, lista de bases pesquisadas, idioma do estudo e, palavras-chave e seus termos relacionados. A seguir, detalhes dessa estratégia são apresentados:

1. **Critério de seleção.** Os seguintes critérios foram utilizados para selecionar as bases de pesquisa deste mapeamento (DIESTE; GRIMáN; JURISTO, 2009): (i) frequência de atualização de conteúdo; (ii) disponibilidade de textos na íntegra; (iii) qualidade dos resultados apresentados pela base; e (iv) versatilidade para exportar os dados obtidos;

2. **Lista de bases pesquisadas.** As bases de pesquisa selecionadas para o mapeamento são apresentadas na Tabela 6. Segundo Kitchenham e Charters (2007) e Petersen, Vakkalanka e Kuzniarz (2015), tais bases atendem aos critérios de seleção de bases de pesquisa supracitados e são consideradas suficientes para a condução desse tipo de estudo para a área de **engenharia de software**.

Tabela 6 – Lista de bases de pesquisa

Base de pesquisa	**Endereço**
ACM Digital Library	dl.acm.org
IEEE Xplore	www.ieeexplore.ieee.org
ScienceDirect	www.sciencedirect.com
Scopus	www.scopus.com
Springer Link	link.springer.com
Web of Science	www.isiknowledge.com

3. **Idioma dos estudos.** Apenas estudos primários escritos em inglês devem ser considerados neste SMS. Essa restrição deve-se ao fato desse idioma ser o mais comum em publicações científicas, além de auxiliar em futuras replicações deste trabalho.

4. **Palavras-chave:** "self-protecting" "learning techniques", com os seguintes sinônimos:

   a) ***self-protecting:*** *"adaptive security", "autonomous security", "self-protecting", "self-protection", "self-safety", "self-securing", "self-security"*;

   b) ***learning techniques:*** *"analytics", "big data", "computational intelligence", "data mining", "data science", "KDD", "learning", "natural language processing", "reinforcement", "semi-supervised", "supervised", "unsupervised"*;

5. ***String* de pesquisa:** O operador booleano OR foi utilizado para relacionar os termos principais e seus respectivos sinônimos. Todos esses termos foram combinados utilizando o operador booleano AND. Portanto, a *string* de busca final é mostrada na Tabela 7.

Tabela 7 – *String* de pesquisa

```
({adaptive security} OR {autonomous security} OR
{self-protecting} OR {self-protection} OR {self-safety} OR
{self-securing} OR {self-security})
AND
({analytics} OR {big data} OR {computational intelligence} OR
{data mining} OR {data science} OR {KDD} OR {learning} OR
{natural language processing} OR {reinforcement} OR
{semi-supervised} OR {supervised} OR {unsupervised})
```

Definir os Critérios de Inclusão (CI) e Critérios de Exclusão (EC) para um mapeamento é um importante elemento para condução e execução do mesmo. A partir da definição desses critérios é possível incluir estudos primários relevantes para responder as QPs e excluir estudos que não permitem extrair respostas para as mesmas. Para este SMS, o seguinte IC foi definido:

- **CI 1:** O estudo primário apresenta uma solução para SApps.

Em contrapartida, os CEs definidos para este mapeamento foram:

- **CE 1:** O estudo não evidencia uma solução para SApps;
- **CE 2:** O estudo é um editorial, artigo de opinião, palestra, opinião, tutorial, poster ou painel;
- **CE 3:** O estudo foi escrito em um idioma diferente do definido pela pesquisa;
- **CE 4:** O estudo possui apenas um resumo ou não está disponível para ser lido na íntegra;
- **CE 5:** O estudo é um trabalho desenvolvido anteriormente pelo mesmo autor. Nesse caso, apenas a versão mais recente do estudo ou a mais completa será considerada.
- **CE 6:** O estudo é um trabalho secundário.

Por fim, vale destacar que nenhum critério de qualidade será aplicado ao mapeamento devido ao número de pesquisadores envolvidos. Para isso, a *string* de busca (Tabela 7) será customizada e executada em cada uma das bases de pesquisa explorando os seguintes campos de busca: título, resumo e palavras-chave. Adicionalmente, o número de estudos resultantes para cada meio de publicação será registrado após a seleção primária dos estudos com base no título e resumo. Por fim, o número de estudos finalmente selecionado de cada base de pesquisa também será registrado. A partir do conjunto de estudos primários selecionados, o título e o resumo de cada estudo será lido e avaliado, conforme os CI e CEs definidos nesta seção. Quando necessário, a introdução e a conclusão também devem ser lidas. Dúvidas na aplicação dos critérios serão resolvidas por meio de mediações com o orientador deste projeto de mestrado acadêmico.

A.4 Extração e síntese de dados

Esta fase resultará em um conjunto de dados que descreve cada estudo primário identificado na Seção A.3. Conforme sugerido por Petersen, Vakkalanka e Kuzniarz (2015), um formulário para cada estudo primário deve ser preenchido, visando facilitar as tarefas de extração e síntese de dados. Com base nos campos propostos pelos autores supracitados, um formulário específico para este mapeamento foi elaborado, como mostra a Tabela 8. Alguns dos tópicos de pesquisa desse formulário serão utilizados posteriormente para responder as questões de pesquisa estabelecidas na Seção A.2.

Tabela 8 – Formulário de extração de dados

Tópico de Pesquisa	**Valor**	**QP**
ID do estudo	Valor inteiro	—
Título	Título do artigo	—
Nome do autor	Nomes dos autores	—
Ano	Ano de publicação	—
Base de pesquisa	Nome da base de pesquisa que recuperou o artigo	—
Solução de autoproteção	*Frameworks*, bibliotecas, ferramentas, *plugins*, arquiteturas, modelos, abordagens, plataformas de software, entre outros	QP 1
Domínio de aplicação	Domínio em que a solução foi aplicada	QP 2
Atributos de qualidade	Atributos de qualidade listados	QP 3
Métodos de avaliação	Estudos de caso, provas de conceito, experimentos, entre outros	QP 4
Maturidade	Nível de maturidade (indústria, academia, mista)	QP 4.1
Recursos de design	Nome da técnica utilizada	QP 5
Modelos arquiteturais e Camadas	Nome dos modelos e camadas	QP 5.1
Estratégias de autoproteção	Nomes das estratégias (reativo, proativo, ou ambos)	QP 6
Dimensões de segurança	Controle de acesso, autenticação, não repúdio, confidencialidade de dados, segurança de comunicação, integridade de dados, disponibilidade, privacidade, autorização, entre outros	QP 7
Tipo de modificação	Estrutural, comportamental, arquitetural, contexto, mista, entre outros.	QP 8
Técnicas de aprendizado, algoritmos e abordagens de aprendizado	Nome das técnicas de aprendizado, nome dos algoritmos e tipos de abordagens (supervisionada, semi-supervisionada, não-supervisionada, por reforço, entre outras)	QP 9

A.5 Limitações

Nesta seção são apresentadas as possíveis limitações que um trabalho dessa natureza (SMS) pode apresentar, a saber:

1. **Omissão de artigos.** O mapeamento utilizará um processo de pesquisa automatizado, sendo a *string* de busca adaptada e processada pelo mecanismo de busca/pesquisa de cada base. Conforme reportado na Seção A.3, seis bases de pesquisa para a obtenção das publicações serão utilizadas e, apesar dos esforços em tornar este trabalho mais confiável,

não é possível afirmar que todos os estudos primários relacionados a temática deste SMS poderão ser recuperados. Entretanto, com o intuito de calibrar a *string* final, pesquisas com diferentes variações deverão ser realizadas para validar se estudos que deveriam ter sido recuperados foram retornados. O caso oposto também deve ser validado, ou seja, estudos que não devem ser recuperados. Em seguida, a *string* final será posteriormente adaptada para os padrões de cada uma das bases de pesquisa. Portanto, acredita-se que dessa maneira nenhum estudo relevante seja excluído do mapeamento.

2. **Confiabilidade dos pesquisadores.** Todos os envolvidos no mapeamento são pesquisadores da área de Engenharia de Software. Nenhum dos estudos incluídos neste mapeamento são de autoria dos membros do grupo de pesquisa ou de pesquisadores relacionados com os autores. Sendo assim, acredita-se que os resultados apresentados neste SMS devam apresentar o atual estado da arte da área pesquisada sem a introdução involuntária de viés.

3. **Parcialidade.** Embora um conjunto de diretrizes seja definido com a finalidade de garantir que o mapeamento seja imparcial, não é possível garantir que toda a informação coletada seja completamente imparcial, pois esta sujeita a interpretação da pessoa que está conduzindo o mapeamento.

4. **Extração de dados.** Outra eventual limitação deste mapeamento está relacionada ao processo de extração de dados. As informações utilizadas para responder as QPs podem nem sempre estar presentes de maneira explícita no texto e algumas informações devem ser interpretadas. Entretanto, quando houver dúvidas/divergências a respeito de um estudo, o mesmo deve ser discutido entre os pesquisadores envolvidos a fim de estabelecer um consenso quanto ao critério a ser aplicado e/ou dado a ser extraído. Adicionalmente, vale destacar que um formulário de pesquisa foi adotado com o objetivo de auxiliar na tarefa de coleta e síntese dos dados, produzindo um arquivo padronizado de dados.

5. **Avaliação da qualidade.** Como o objetivo deste mapeamento é identificar soluções de autoproteção para SApps, a qualidade dos estudos não será avaliada. Entretanto, deve-se reconhecer a importância da avaliação de qualidade e esse passo poderá ser considerado em uma futura versão deste mapeamento.

6. **Replicabilidade.** Em função das bases de pesquisa utilizadas estarem disponíveis em ambiente virtual e serem constantemente atualizadas, é possível que em uma futura pesquisa os resultados não sejam exatamente iguais, o que poderia dificultar a reprodução dos dados coletados a partir deste SMS.

Por fim, vale mencionar que um único pesquisador conduzirá as etapas de extração e síntese de dados. Embora um conjunto de diretrizes tenha sido definido com a finalidade de

garantir que o mapeamento seja imparcial, não é possível garantir que toda a informação coletada seja completamente imparcial. Para o futuro espera-se possuir ao menos duas pessoas envolvidas nessas etapas.

A.6 Síntese dos resultados

Ao reportar os resultados deste mapeamento, as seguintes atividades devem ser realizadas: (i) discussão sobre os resultados (ou seja, descrição de cada estudo primário e detalhes de qualquer meta-análise que tenha sido realizada); (ii) discussão das principais descobertas, forças e fraquezas do mapeamento e os significados dessas descobertas como, por exemplo, aplicabilidade, benefícios, efeitos adversos e riscos; (iii) conclusão, incluindo recomendações como, por exemplo, possíveis implicações práticas para o desenvolvimento de software e para a área de pesquisa e questões de pesquisa não respondidas; e, por fim, (iv) elabora uma discussão sobre as limitações do mapeamento.

APÊNDICE B – LISTA DE ESTUDOS PRIMÁRIOS

Na Tabela 9 é apresentada a lista de estudos primários ordenada pelo título resultante da condução do mapeamento sistemático reportado no Capítulo 3.

Tabela 9 – Lista final de estudos selecionados para a extração e síntese de dados

#	Referência
E1	Hyun Jung La and Soo Dong Kim. **A machine learning framework for adaptive fintech security provisioning**. *Journal of Internet Technology*, 19(5):1545–1553, SEP 2018. 18th International Conference on Information and Communications Security (ICICS), Singapore, SINGAPORE, NOV 29-DEC 02, 2016.
E2	S. Iannucci, O. D. Barba, V. Cardellini, and I. Banicescu. **A performance evaluation of deep reinforcement learning for model-based intrusion response**. In *2019 IEEE 4th International Workshops on Foundations and Applications of Self* Systems (FAS*W)*, pages 158–163, June 2019.
E3	C. Benzaid and T. Taleb. **Ai for beyond 5g networks: A cyber-security defense or offense enabler?** *IEEE Network*, 34(6):140–147, November 2020.
E4	S. Boudko and H. Abie. **Adaptive cybersecurity framework for healthcare internet of things**. In *2019 13th International Symposium on Medical Information and Communication Technology (ISMICT)*, pages 1–6, May 2019.
E5	Mohamed Ibrahim Beer and Mohd Fadzil Hassan. **Adaptive security architecture for protecting restful web services in enterprise computing environment**. *Service Oriented Computing and Applications*, 12(2):111, 2018.
E6	U. Tariq, A. O. Aseeri, M. S. Alkatheiri, and Y. Zhuang. **Context-aware autonomous security assertion for industrial iot**. *IEEE Access*, 8:191785–191794, 2020.
E7	Donald M. Allen and Dmitry Goloubew. **Customer self-remediation of proactive network issue detection and notification**. In Helmut Degen and Lauren Reinerman-Jones, editors, *Artificial Intelligence in HCI*, pages 197–210, Cham, 2020. Springer International Publishing.

Continua na próxima página

Continuando da página anterior

#	Referência
E8	Z. Zhou, X. Kuang, L. Sun, L. Zhong, and C. Xu. **Endogenous security defense against deductive attack: When artificial intelligence meets active defense for online service**. *IEEE Communications Magazine*, 58(6):58–64, June 2020.
E9	A.K. Sarica and P. Angin. **Explainable security in sdn-based iot networks**. *Sensors (Switzerland)*, 20(24):1–30, 2020.
E10	Chad L. Calvert and Taghi M. Khoshgoftaar. **Impact of class distribution on the detection of slow http dos attacks using big data**. *Journal of Big Data*, 6(1):67, 2019.
E11	Jun Yang, Mengyu Zhou, and Baojiang Cui. **Mlab-bilstm: Online web attack detection via attention-based deep neural networks**. In Shui Yu, Peter Mueller, and Jiangbo Qian, editors, *Security and Privacy in Digital Economy*, pages 482–492, Singapore, 2020. Springer Singapore.
E12	A. Kaci and A. Rachedi. **Mc-track: A cloud based data oriented vehicular tracking system with adaptive security**. In *2019 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6, Dec 2019.
E13	Lianbing Deng, Daming Li, Xiang Yao, David Cox, and Haoxiang Wang. **Mobile network intrusion detection for iot system based on transfer learning algorithm**. *Cluster Computing*, 22(4):9889, 2019.
E14	T. T. Tun, M. Yang, A. K. Bandara, Y. Yu, A. Nhlabatsi, N. Khan, K. M. Khan, and B. Nuseibeh. **Requirements and specifications for adaptive security: Concepts and analysis**. In *2018 IEEE/ACM 13th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 161–171, May 2018.
E15	C. Benzaïd, M. Boukhalfa, and T. Taleb. **Robust self-protection against application-layer (d)dos attacks in sdn environment**. In *2020 IEEE Wireless Communications and Networking Conference (WCNC)*, pages 1–6, May 2020.
E16	Francesco Restuccia, Salvatore D'Oro, Liyang Zhang, and Tommaso Melodia. ***The Role of Machine Learning and Radio Reconfigurability in the Quest for Wireless Security***, pages 191–221. Springer International Publishing, Cham, 2019.

Continua na próxima página

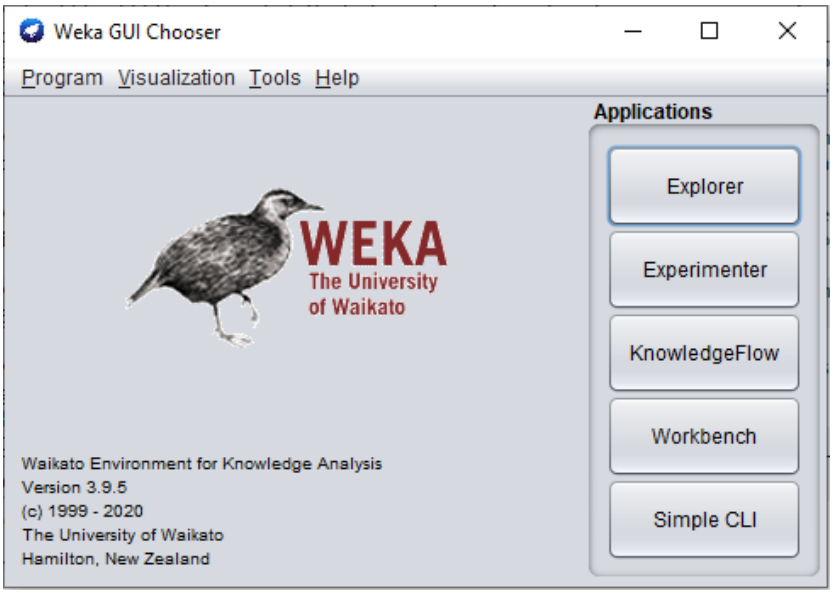
Continuando da página anterior

#	Referência
E17	Salam Khanji and Asad Khattak. **Towards a novel intrusion detection architecture using artificial intelligence**. In *Proceedings of the 2020 9th International Conference on Software and Information Engineering (ICSIE)*, ICSIE 2020, pages 185–189, New York, NY, USA, 2020. Association for Computing Machinery.

APÊNDICE C – PROCESSO DE MINERAÇÃO BASEADO NA FERRAMENTA WEKA

O *workbench* Weka disponibiliza, por meio de interfaces gráficas, funcionalidades para preparação de dados, seleção de atributos, treinamento de modelos de classificação, associação, clusterização e regressão, e análise e visualização dos modelos de treinamento. Além disso, esse *workbench* disponibiliza uma API para linguagem Java, que permite codificar as funcionalidades disponíveis em suas interfaces gráficas em aplicações customizadas. Para isso, são disponibilizados quatro ambientes de trabalho que são acessados a partir de uma tela inicial conforme é ilustrado na Figura 58. A seguir, uma descrição de cada ambiente é apresentada.

Figura 58 – WEKA - Tela principal



Fonte: Autoria própria

`Explorer`. Viabiliza de modo interativo o carregamento de conjuntos de dados (*datasets*) em diversos formatos e o carregamento a partir de bancos de dados relacionais. Além disso, permite aplicar filtros em instâncias e atributos do *dataset* carregado, aplicar algoritmos de seleção de atributos, desenvolver modelos de treinamento a partir de diversos algoritmos e técnicas de aprendizado (por exemplo, classificação, associação, clusterização e regressão), e visualizar a performance dos modelos de treinamento.

Experimenter. Permite avaliar e comparar de modo automatizado, a performance de vários modelos de treinamento a partir de diferentes variáveis e perspectivas como, por exemplo, algoritmos de treinamento, aplicação de filtros e hiperparâmetros.

KnowledgeFlow. Este ambiente permite criar um fluxo de conhecimento a partir de blocos lógicos que são interconectados por flechas direcionais, que representam o fluxo de conhecimento do processo de treinamento. Cabe destacar que todas as funcionalidades disponíveis no ambiente `Explorer` estão disponíveis neste ambiente para uma modelagem e execução visual.

Workbench. São disponibilizadas em um mesmo ambiente a integração dos ambientes `Explorer`, `Experimenter`, e `KnowledgeFlow`, onde é permitido ao usuário customizar os objetos, interfaces e menus que são exibidos.

Simple CLI. É um ambiente de terminal que permite utilizar os mesmos recursos disponíveis nas interfaces gráficas por meio de linhas de comando.

Para as tarefas de preparação de dados, treinamento e análise que compõem a abordagem de autoproteção proposta por este trabalho (veja a Seção 4.1.2) foi adotado o ambiente `Workbench` do WEKA. Neste contexto, para descrever o processo adotado, nas seções de Seção C.1 a Seção C.3 será considerado o *dataset* das *features* de SQLi, que é resultante da atividade de mineração de dados do estudo de caso deste trabalho (veja Seção 5.3).

C.1 Preparação dos dados

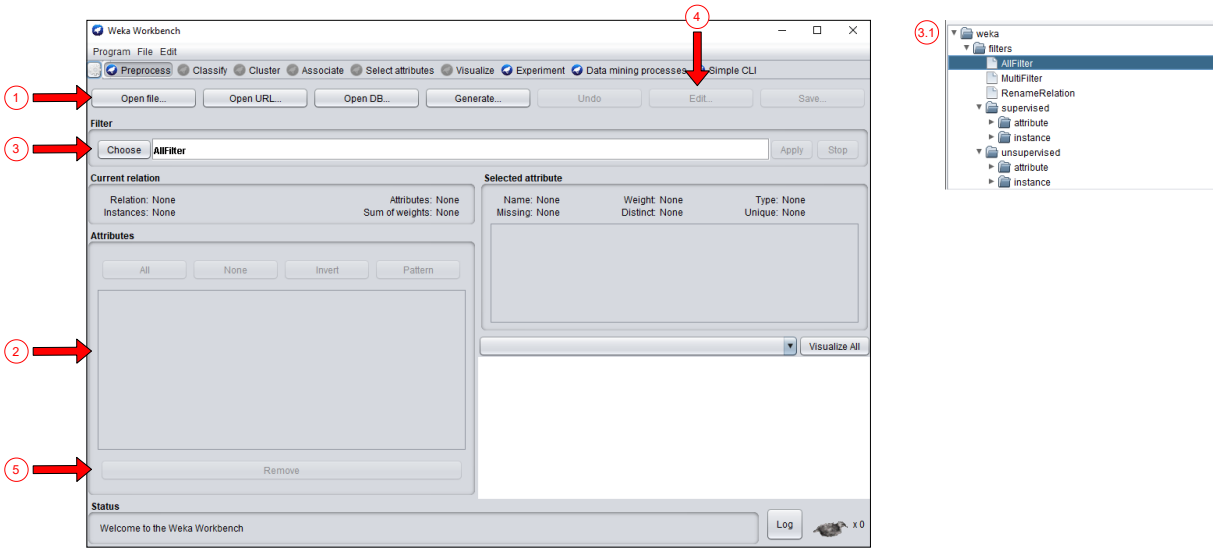
Para a atividade de preparação do *dataset* SQLi foram adotadas as interfaces `Preprocess`, `Classify` e `Select Attributes` do ambiente *Workbench*. A Figura 59 ilustra cada uma das tarefas executadas na primeira interface, as quais são descritas em detalhe a seguir.

Tarefa 1. Nesta tarefa é possível carregar o *dataset* SQLi em formato `CSV`, sendo que outros formatos estão disponíveis. Após carregar o *dataset*, objetos dispostos na interface são habilitados.

Tarefa 2. Ao carregar os *datasets* na área *Attributes* são apresentados os atributos disponíveis. Em seguida, os atributos são avaliados, onde a aplicação de filtros e remoção de atributos são planejados.

Tarefa 3 Nesta tarefa, filtros são aplicados nos atributos ou instâncias do *dataset* conforme planejado na **Tarefa 2**. A **Tarefa 3.1** exemplifica os filtros disponíveis, onde são categorizados

Figura 59 – Workbench - Preprocess



Fonte: Autoria própria

em *supervised* e *unsupervised*. Ao escolher os filtros contidos na categoria *supervised* é importante atentar-se aos parâmetros disponíveis, para que se tenham resultados eficientes (WITTEN *et al.*, 2011).

Tarefa 4. Esta atividade pode ocorrer sempre que o responsável pela preparação dos dados queira visualizar os dados disponíveis no *dataset*. Além disso, dados são podem ser modificados e instâncias podem ser adicionadas manualmente.

Tarefa 5. Nesta tarefa é possível remover atributos do *dataset*, embora seja possível efetuar a mesma tarefa por meio de filtros especializados, que estão disponíveis na **Tarefa 3**.

Após carregar o *dataset* SQLi em formato `CSV`, alguns filtros, de acordo com as boas práticas citadas por Witten *et al.* (2011), foram aplicados. A seguir, detalhes de tais filtros são reportados:

- **`unsupervised/attribute/ClassAssigner`**. este filtro é aplicado ao atributo que será definido como classe do *dataset*, ou seja, aquele que será responsável por apresentar a classificação da instância. Neste contexto, o atributo selecionado foi `attack`.
- **`unsupervised/attribute/NumericToNominal`**. o atributo `attack` pode conter os valores numéricos "1" para representação de ataques ou "`0`" para requisição normal. Para que seja possível criar o modelo de classificação, este filtro foi aplicado para converter o atributo `attack` de numérico para nominal.
- **`unsupervised/attribute/Remove`**. o *dataset* SQLi possui os atributos *Source* e *Id*, onde o primeiro aponta qual o *dataset* de comportamento de origem da instância do *dataset*

SQLi, e o segundo descreve o identificador da instância na origem. Esses atributos podem perturbar o modelo de treinamento por não terem qualquer relação com as situações de ataques ou acessos normais. Por esse motivo, esse filtro foi aplicado para remover ambos os atributos do processo de treinamento.

- **unsupervised/instance/RemoveDuplicates**. este filtro foi aplicado para remover instâncias repetidas do `dataset` SQLi.
- **unsupervised/instance/Randomize**. este filtro foi aplicado para embaralhar aleatoriamente a ordem das instâncias. Esta atividade possibilita maior probabilidade de generalização e precisão do modelo de treinamento.
- **supervised/instance/ClassBalancer**. ao observar os valores da classe `attacK` foi constatado que 35.478 (86,18%) instâncias representavam requisições normais, enquanto 5.691 (13,82%) instâncias representam ataques. Essa desigualdade dos valores da classe `attack` pode gerar modelos de treinamento tendenciosos. Portanto, este filtro foi aplicado para buscar aproximar os valores. Cabe destacar que o parâmetro *Number of discretization intervals* não foi alterado, ficando com o seu valor padrão (10).

É importante ressaltar que durante o processo de aplicação de filtros foi utilizada a interface `Classify` para avaliar a performance de treinamento com os dados do *dataset* antes e depois da aplicação do filtro `supervised/instance/ClassBalancer`. Neste sentido, foi adotado o algoritmo `J48` com validação cruzada de 10 grupos. Após as avaliações, foi observado que o tempo de treinamento do conjunto balanceado em relação ao conjunto desbalanceado aumenta em torno de 20%. Além disso, a taxa de instâncias classificadas corretamente cai de 97,56% para 93,20%. Portanto, diante dos dados resultantes, o filtro `supervised/instance/ClassBalancer` não foi adotado no conjunto de dados final. A Figura 60 ilustra cada uma das tarefas executadas na interface `Classify`, as quais são detalhadas a seguir.

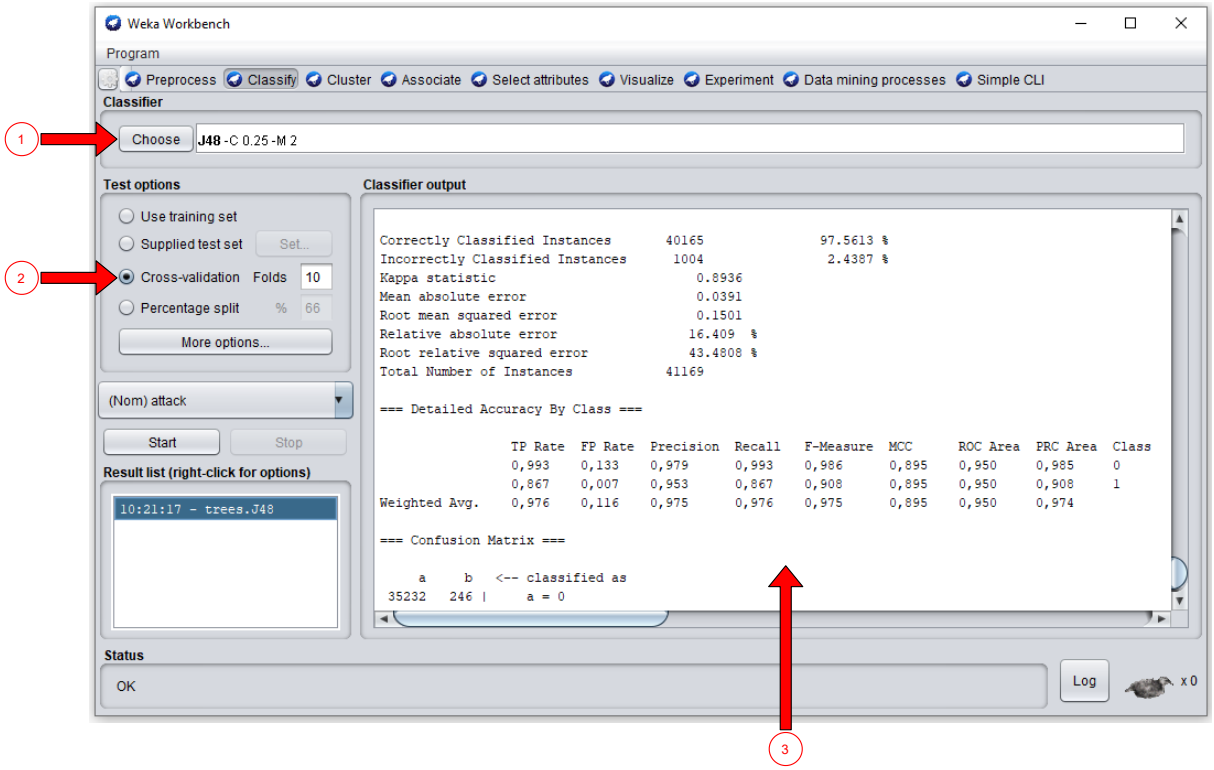
Tarefa 1. Nesta tarefa é selecionado o algoritmo de treinamento, onde são classificados em: *bayes*, *functions*, *lazy*, *meta*, *misc*, *rules* e *trees*.

Tarefa 2. Foi selecionado nesta tarefa o tipo de amostragem dos dados para o processo de treinamento, onde foi selecionada a opção `Cross-validation Folds` com valor 10.

Tarefa 3. Após o término do processo de treinamento, na janela `Classifier output`, podem ser interpretados e avaliados os resultados.

Após aplicar os filtros supracitados, a interface `Select Attributes` foi utilizada para a tarefa de seleção de atributos. De acordo com Brownlee (2021), com essa tarefa é possível

Figura 60 – Workbench - Classify



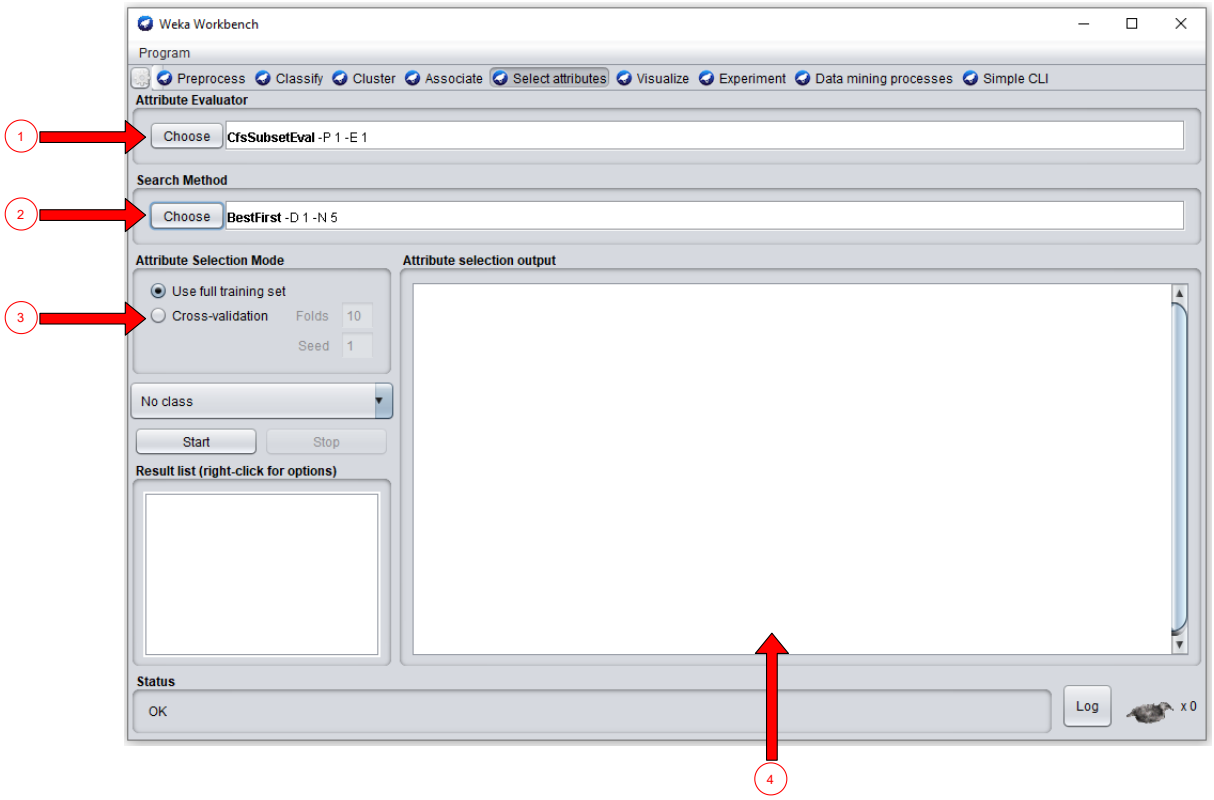
Fonte: Autoria própria

selecionar um conjunto de atributos que sejam relevantes para o processo de treinamento, criando modelos precisos, menos complexos e exigindo menos dados (instâncias). Além disso, vale destacar que a não execução deste processo pode deixar perturbações no modelo de treinamento que podem causar sobre ajuste *overfitting*, ou seja, o modelo pode ser ineficaz em classificar dados diferentes do `dataset`. Após o processo de extração de atributos, foi executado novamente o processo de treinamento (conforme ilustrado na Figura 60) para os *datasets* original e com extração de atributos. Ao final foi observado que houve um ganho de tempo no processo de treinamento para o *dataset* com extração de atributo (27 segundos) em relação ao *dataset* sem extração (2 minutos). Entretanto, a taxa de verdadeiro-positivo (do inglês, True-Positive - TP) é maior (86,7%) para os dados sem extração, se comparada aos dados com extração (76,9%). Portanto, tomou-se a decisão de não aplicação do processo de extração de atributos. A Figura 61 ilustra cada uma das tarefas executadas na interface `Select Attributes`, as quais são detalhadas a seguir.

Tarefa 1. Nesta tarefa deve ser escolhido o avaliador de atributo, onde foi selecionado o item `CfsSubsetEval`. Este avaliador seleciona um subconjunto de atributos baseado em correlação de aprendizado.

Tarefa 2. Baseado no avaliador de atributo selecionado, deve ser escolhido o método de busca,

Figura 61 – Workbench - Select Attribute



Fonte: Autoria própria

onde foi selecionado o método `BestFirst`.

Tarefa 3. Nesta tarefa foi definido o modo de seleção de atributo `User full training set`.

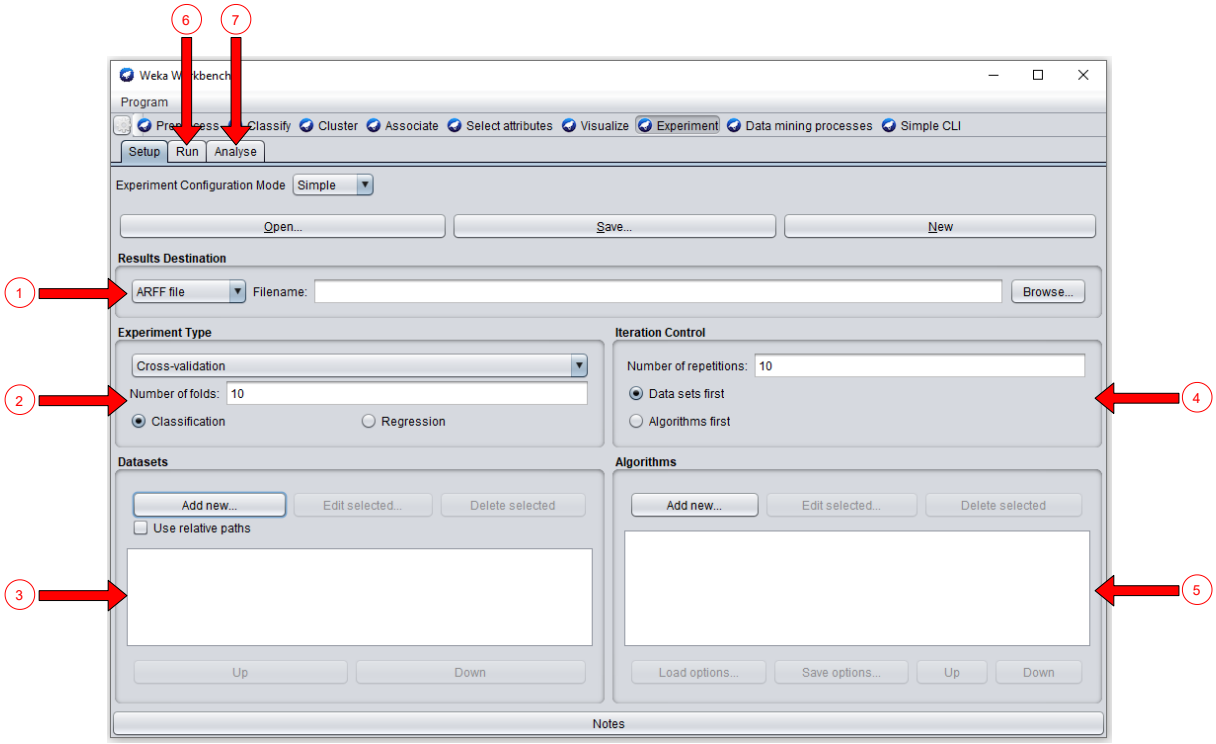
Tarefa 4. Na janela `Attribute selection output` é apresentado o resultado do processamento da tarefa de seleção de atributos.

C.2 Treinamento e avaliação dos modelos de classificação

Para a execução desta atividade foi utilizada a interface `Experiment`, onde é possível selecionar diferentes algoritmos para avaliar a melhor performance sobre um determinado *dataset*. Para isso, foi utilizado o *dataset* resultante da atividade de preparação de dados (veja Seção C.1) com os seguintes algoritmos: `Logistic`, `LibSVM`, `AdaBoostM1`, `J48`, `K-Nearest Neighbor` (IBk-K) e `RandomForest`. A Figura 62 ilustra cada uma das tarefas executadas na interface `Experiment`, as quais são detalhadas a seguir.

Tarefa 1. Esta tarefa visa produzir a avaliação de performance com vários algoritmos, os dados resultantes são armazenados em um arquivo em formato `ARFF` (do inglês, *Attribute-Relation*

Figura 62 – Workbench - Experiment



Fonte: Autoria própria

File Format). Nessa tarefa é possível selecionar o local onde será salvo e o nome do arquivo.

Tarefa 2. Nesta tarefa deve ser selecionado o tipo de treinamento, podendo ser `Classification` ou `Regression`. Além disso, deve ser escolhido o modo de uso do *dataset* para o processo de treinamento. Para a avaliação dos algoritmos supracitados foi selecionado o tipo de treinamento `Classification` e validação cruzada (`Cross-validation`) de 10 grupos para o `dataset`.

Tarefa 3. Deve ser selecionado nesta tarefa o(s) *dataset(s)* que participarão da avaliação, sendo utilizado aqui o *dataset* resultante da atividade de preparação de dados (veja Seção C.1).

Tarefa 4. Nesta tarefa foi selecionado a opção `Data set first` com valor 10 no campo *Number of repetitions*, o que significa que os grupos de *datasets* selecionados na **Tarefa 2** sofreram 10 variações. No exemplo ilustrado na Figura 62, o classificador atuará em 100 subconjuntos de *datasets* diferentes.

Tarefa 5. Foram selecionados nesta tarefa os algoritmos de treinamento supracitados.

Tarefa 6. Esta tarefa permite iniciar o processamento da atividade de avaliação.

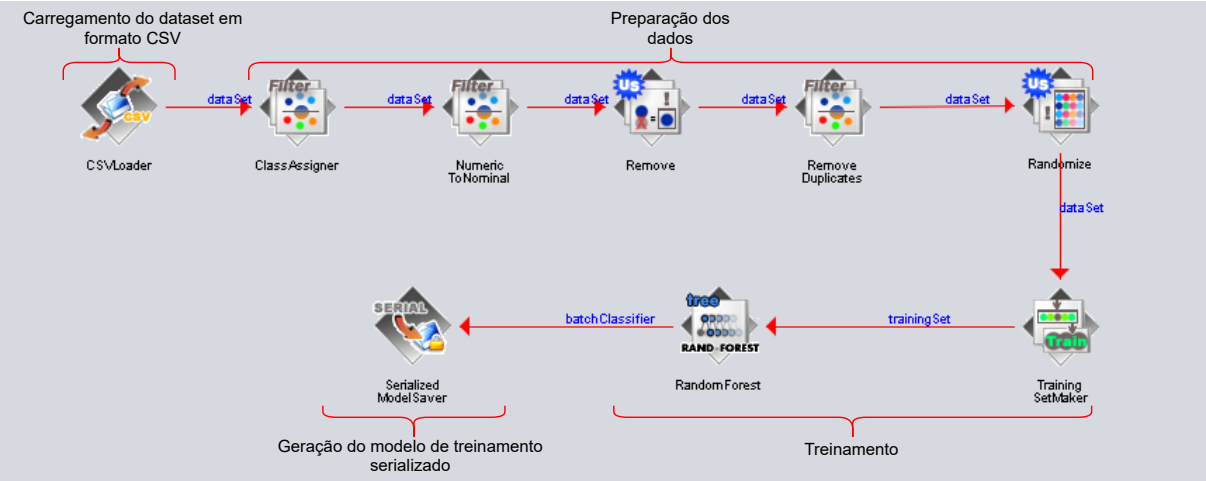
Tarefa 7. Após o processamento iniciado pela **Tarefa 6**, esta tarefa permite avaliar os dados resultantes.

Ao analisar os dados resultantes da avaliação de treinamento, foram também avaliados os dados resultantes (Tarefa 7). Para isso, foram utilizadas duas métricas de avaliação, sendo "Porcentagem de avaliações corretas" e "Curva ROC". Em ambas as métricas, o algoritmo *Random Forest* obteve a melhor performance.

C.3 Fluxos de conhecimento

Nesta atividade representa um fluxo de conhecimento, que é capaz de reproduzir fielmente as atividades descritas nas seções C.1 e C.2. Esse fluxo pode ser utilizado para a geração de modelos de classificação atualizados a partir de novos dados de *features*. Conforme ilustrado na Figura 63, o objeto `CSVLoader` é responsável por carregar o arquivo do *dataset* de *feature* em formato `CSV`. Os objetos `ClassAssigner`, `NumericToNominal`, `Remove`, `RemoveDuplicates` e `Randomize` são equivalentes aos filtros apresentados na Seção C.1. Os objetos `TrainingSetMaker` e `Rand Forest` são responsáveis pelo treinamento do modelo de classificação utilizando o algoritmo *Random Forest*. Por fim, o objeto `ArffSaver` é responsável por gerar o modelo de classificação serializado, que será incorporado no *loop* MAPE correspondente.

Figura 63 – Workbench - KnowledgeFlow



Fonte: Autoria própria