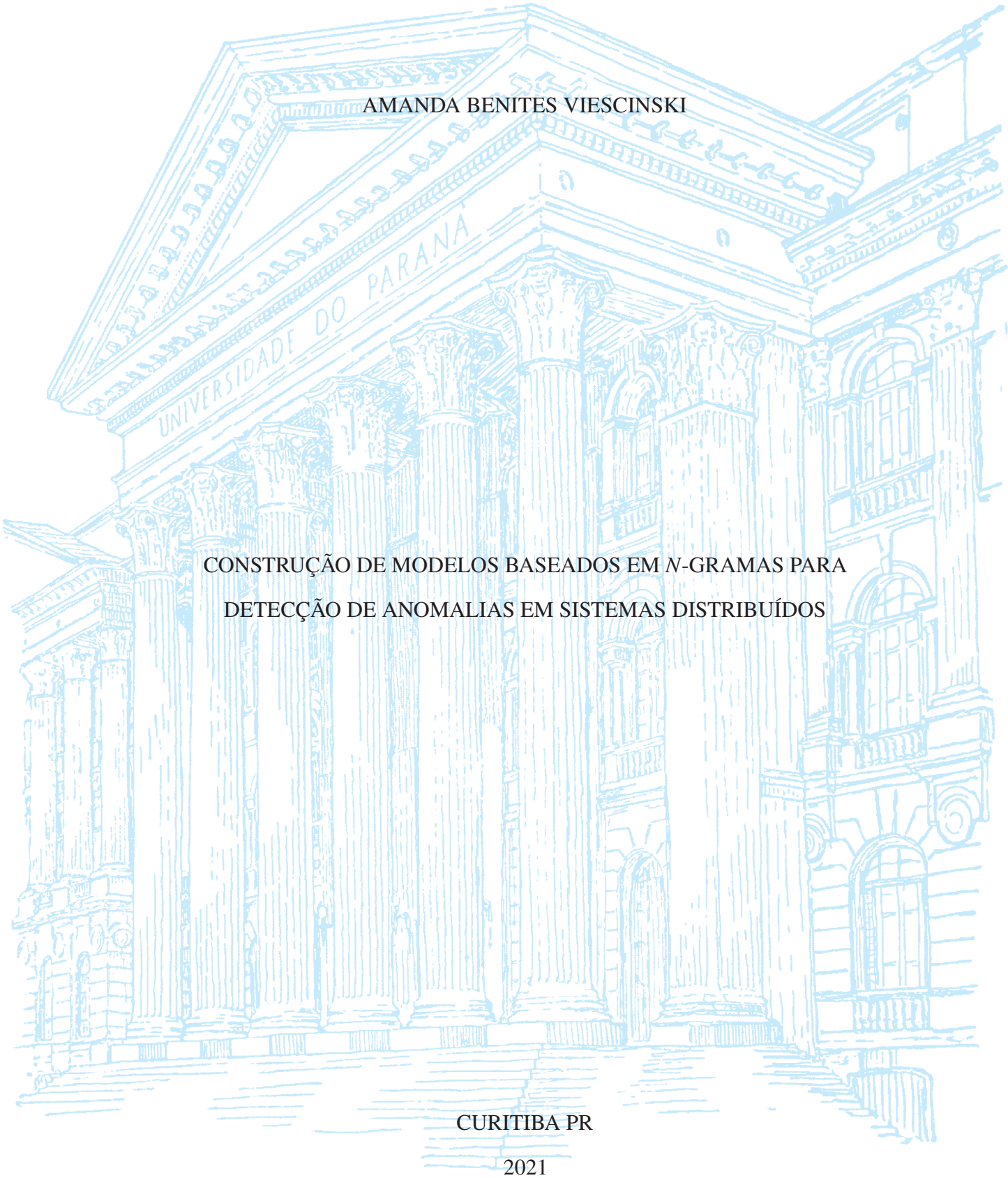
UNIVERSIDADE FEDERAL DO PARANÁ

AMANDA BENITES VIESCINSKI



CONSTRUÇÃO DE MODELOS BASEADOS EM *N*-GRAMAS PARA DETECÇÃO DE ANOMALIAS EM SISTEMAS DISTRIBUÍDOS

CURITIBA PR

2021

AMANDA BENITES VIESCINSKI

CONSTRUÇÃO DE MODELOS BASEADOS EM *N*-GRAMAS PARA DETECÇÃO DE ANOMALIAS EM SISTEMAS DISTRIBUÍDOS

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre em Informática no Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, da Universidade Federal do Paraná.

Área de concentração: *Ciência da Computação*.

Orientador: Carlos Alberto Maziero.

CURITIBA PR

2021

Catalogação na Fonte: Sistema de Bibliotecas, UFPR
Biblioteca de Ciência e Tecnologia

V665c Viescinski, Amanda Benites
Construção de modelos baseados em *n*-gramas para detecção de anomalias em sistemas distribuídos [recurso eletrônico] / Amanda Benites Viescinski – Curitiba, 2021.

Dissertação - Universidade Federal do Paraná, Setor de Ciências Exatas, Programa de Pós-graduação em Informática.

Orientador: Carlos Alberto Maziero.

1. Redes de computação – Segurança. 2. Sistemas de detecção de intrusão (modelos). I. Universidade Federal do Paraná. II. Maziero, Carlos Alberto. III. Título.

CDD: 005.8

Bibliotecária: Roseny Rivelini Morciani CRB-9/1585

TERMO DE APROVAÇÃO

Os membros da Banca Examinadora designada pelo Colegiado do Programa de Pós-Graduação em INFORMÁTICA da Universidade Federal do Paraná foram convocados para realizar a arguição da dissertação de Mestrado de **AMANDA BENITES VIESCINSKI** intitulada: **Construção de Modelos Baseados em N-Gramas para Detecção de Anomalias em Sistemas Distribuídos**, sob orientação do Prof. Dr. CARLOS ALBERTO MAZIERO, que após terem inquirido a aluna e realizada a avaliação do trabalho, são de parecer pela sua APROVAÇÃO no rito de defesa.

A outorga do título de mestre está sujeita à homologação pelo colegiado, ao atendimento de todas as indicações e correções solicitadas pela banca e ao pleno atendimento das demandas regimentais do Programa de Pós-Graduação.

CURITIBA, 22 de Março de 2021.

Assinatura Eletrônica
22/03/2021 15:24:49.0
CARLOS ALBERTO MAZIERO
Presidente da Banca Examinadora (UNIVERSIDADE FEDERAL DO PARANÁ)

Assinatura Eletrônica
22/03/2021 16:05:52.0
LUIZ CARLOS PESSOA ALBINI
Avaliador Interno (UNIVERSIDADE FEDERAL DO PARANÁ)

Assinatura Eletrônica
23/03/2021 12:59:19.0
RAFAEL RODRIGUES OBELHEIRO
Avaliador Externo (UNIVERSIDADE DO ESTADO DE SANTA CATARINA)

Aos meus pais, Gervásio (in memoriam) & Roseli.

AGRADECIMENTOS

Durante o período de realização desta pesquisa de mestrado, convivi com profissionais e pesquisadores ímpares, que contribuíram para o meu crescimento em vários âmbitos. Agradeço primeiramente ao meu orientador, Carlos Maziero, por confiar no meu potencial ao me aceitar como orientanda. Sua orientação, motivação e compreensão das minhas limitações foram essenciais para a conclusão desta pesquisa.

Meus agradecimentos a todos os membros do Laboratório de Redes, Sistemas Distribuídos e Segurança (LaRSiS), pela parceria e apoio durante este período. Em especial a minha equipe de pesquisa, Tiago Heinrich e Newton Carlos Will, espero que possamos produzir muitas pesquisas juntos futuramente. Agradeço também ao David Lanoë e toda a equipe do laboratório *CentraleSupélec*, por cederem a base de dados que possibilitou esta pesquisa.

Minha espiritualidade serviu como refúgio nos momentos difíceis, principalmente por enfrentarmos uma pandemia neste momento. Sendo assim, agradeço a Deus por tudo que me proporcionou viver até aqui, por me manter forte e saudável durante este período para poder completar mais esta etapa.

Sou imensamente grata aos meus pais, de sangue (Roseli e Gervásio) e de coração (Cleudide e Roberto), por me ensinar a importância dos estudos e da persistência. Aos meus irmãos, principalmente ao Danilo Viescinski, por ter me apresentado o mundo da computação e por todos os ensinamentos passados ao longo da vida.

Esta pesquisa pode ser iniciada graças a coragem de uma pessoa: José Wilson. Muito obrigado por sua coragem de sair do seu lugar de conforto e encarar algo totalmente novo ao meu lado. Sua força e determinação me incentivam diariamente a buscar o que sonhamos. Não há palavras para expressar o quanto te admiro e sou grata a você.

Por fim, agradeço a CAPES pelo apoio financeiro através da bolsa de mestrado. Este apoio viabilizou a permanência no curso e tornou possível a dedicação exclusiva a pesquisa, influenciando positivamente no meu desempenho.

RESUMO

A segurança é fundamental em sistemas distribuídos. Contudo, a implementação de sistemas seguros para proteger informações têm sido uma meta dificilmente alcançada. Além dos mecanismos de prevenção, o monitoramento desses sistemas e de seus componentes, no intuito de identificar comportamentos de natureza suspeita, é uma abordagem amplamente adotada. Para isso, sistemas de detecção de intrusão (*Intrusion Detection System* - IDS) são implantados em diversos pontos do sistema monitorado, configurados para supervisionar ações locais, bem como comunicar ocorrências de eventos. Para que um IDS exerça suas funções, é necessário que seja capaz de reconhecer comportamentos benignos e maliciosos. Diferentes técnicas de detecção podem ser aplicadas ao projetar um IDS. A técnica considerada mais simples é a detecção por assinatura, que reconhece as estratégias padrões de uma ameaça. Para isso, é necessária uma base de execuções desses ataques, extraindo suas principais características que resultará em uma assinatura. Sendo assim, apenas ataques previamente conhecidos são detectados. A técnica de detecção por anomalias não requer o conhecimento prévio dos padrões de ataques, já que o comportamento normal do ambiente monitorado é utilizado como base para detecção. Neste caso, um modelo de comportamento normal do sistema é construído e utilizado pelo sistema de detecção para checar desvios no comportamento do ambiente monitorado. Este trabalho propõe uma técnica de modelagem comportamental para aplicações distribuídas através dos traços de operações dos seus nós. A abordagem descrita neste trabalho permite a elaboração de uma estratégia para a identificação de anomalias em um sistema distribuído. A estratégia escolhida consiste na construção de modelos de comportamento normal do sistema sob estudo, que são dispostos em conjuntos de n-gramas de eventos (sequências de n eventos consecutivos envolvendo um ou mais nós). O objetivo da proposta é construir modelos funcionais e eficazes, que possibilitem detecções de anomalias no sistema, com taxas reduzidas de falsos positivos. Toda a investigação é realizada com base em traços de uma aplicação distribuída real. São apresentados os procedimentos realizados para a construção de modelos parciais, incluindo as abordagens utilizadas para a ordenação dos eventos que ocorreram no sistema e o cálculo dos n-gramas. Os resultados obtidos através da avaliação dos modelos destacam a viabilidade do uso de n-gramas para representar atividades corretas de um sistema, com resultados propícios na taxa de falso positivo e também em termos de acurácia.

Palavras-chave: Modelos. Sistemas de detecção de intrusão. Detecção de anomalias.

ABSTRACT

Security is critical in distributed systems. However, the implementation of secure systems to protect sensitive information has been a difficult goal to achieve. In addition to the prevention mechanisms, the monitoring of these systems and their components to identify suspicious behaviors is an adopted approach. For this purpose, Intrusion Detection System (IDS) are deployed at different points of the monitored system, configured to supervise local actions, as well as report event occurrences. For an IDS to perform its functions, it must be able to recognize benign and malicious behavior. Different detection techniques can be applied when designing an IDS. The technique considered simplest is signature detection, which recognizes the standard strategies of a threat. For that, it is necessary to have a database of executions of these attacks, extracting their main characteristics that will result in a signature. Therefore, only previously known attacks are detected. Anomaly detection does not require prior knowledge of attack patterns, as the normal behavior of the monitored environment is used as a basis for detection. In this case, a model of the normal behavior of the system is constructed and used by the detection system to check for deviations in the behavior of the monitored environment. This work proposes a behavioral modeling technique for distributed applications through the traces of operations of its nodes. The approach described in this work allows the development of a strategy for the identification of anomalies in a distributed system. The chosen strategy consists of building models of the normal behavior of the system under study, which is arranged in sets of n-grams of events (sequences of n consecutive events involving one or more nodes). The objective of the work is to build adequate and effective models, which make it possible to detect anomalies in the system, with reduced rates of false positives. All research is carried out based on traces of a real distributed application. The procedures performed for the construction of partial models are presented, including the approaches used for the ordering of the events that occurred in the system and the calculation of the n-grams. The results obtained through the evaluation of the models highlight the feasibility of using n-grams to represent correct activities of a system, with favorable results in the false positive rate and also accuracy.

Keywords: Models. Intrusion Detection System. Anomaly detection.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ACRÔNIMOS

AD	*Anomaly-based Detection*
APAP	*Advanced Payload Analyzer Pre-processor*
CBF	*Counting Bloom Filters*
DAG	*Directed Acyclic Graph*
DFS	*Depth-First Search*
DIR	*Directory Service*
DPI	*Deep Packet Inspection*
FSA	*Finite-State Automaton*
FTP	*File Transfer Protocol*
HIDS	*Host-Based Intrusion Detection System*
HTTP	*HyperText Transfer Protocol*
IDS	*Intrusion Detection System*
IP	*Internet Protocol*
McPAD	*Multiple classifier Payload-based Anomaly Detector*
MRC	*Metadata and Replica Catalog*
NIDS	*Network-Based Intrusion Detection System*
OSD	*Object Storage Device*
PAYL	*Anomalous Payload-Based Intrusion Detection*
PCkAD	*Packet Chunk Anomaly Detector*
SD	*Signature-based Detection*
SVM	*Support Vector Machine*

LISTA DE SÍMBOLOS

$[a,\ b,\ \ldots]$	uma sequência de eventos ordenados.
n	tamanho de uma grama.
T	um traço.
TC	um traço correto.
$\mathbb{E}$	todas as execuções obtidas do sistema que estão presentes na base.
E	uma única execução distribuída presente em $\mathbb{E}$.
$\mathbb{C}$	subconjunto de execuções corretas presentes em $\mathbb{E}$.
$\mathbb{G}$	uma sequência de n eventos consecutivos (n-grama).
$\mathbb{G}(E, n)$	todos os n-gramas distintos de tamanho n obtidos de uma execução E.
$\mathbb{M}$	modelo(s).
$\mathbb{M}\cup$	modelo(s) de união.
$\mathbb{M} \cup (n)$	modelo(s) de união com gramas de tamanho n.
$\mathbb{M}\cap$	modelo(s) de interseção simples.
$\mathbb{M} \cap (n)$	modelo(s) de interseção simples com gramas de tamanho n.
p	quantidade de partições de $\mathbb{C}$.
$\mathbb{P}$	uma partição composta por múltiplas E.
$\mathbb{M}'_i$	modelo parcial de união.
$\mathbb{M}'_i(n)$	modelo parcial de união com gramas de tamanho n.
$\mathbb{M}\sqcap$	modelo(s) de interseção ampla.
$\mathbb{M} \sqcap (n)$	modelo(s) de interseção ampla com gramas de tamanho n.
L	uma lista com os últimos n eventos observados, em ordem de ocorrência.
$\mathbb{A}$	subconjunto de execuções anômalas (ataques) presentes em $\mathbb{E}$.
$\mathbb{P}V_p$	partições de $\mathbb{C}$ para validação.
$\mathbb{M}V$	modelos diversificados de $\mathbb{P}V_p$ para validação.
$\mathbb{M}V_i$	um modelo para validação.
$\mathbb{M}V\cup$	todos os modelos de união, pertencentes a $\mathbb{M}V$ para validação.
$\mathbb{M}V\cap$	todos os modelos de interseção simples, pertencentes a $\mathbb{M}V$ para validação.
$\mathbb{M}V\sqcap$	todos os modelos de interseção amplo, pertencentes a $\mathbb{M}V$ para validação.
$\mathbb{M}V\cup_i$	um modelo de união para validação.
$\mathbb{M}V\cap_i$	um modelo de interseção simples para validação.
$\mathbb{M}V\sqcap_i$	um modelo de interseção amplo para validação.

SUMÁRIO

1 INTRODUÇÃO

Um sistema distribuído é uma coleção de entidades independentes que cooperam para resolver um problema que não pode ser resolvido individualmente (Kshemkalyani e Singhal, 2011). Consequentemente, sistemas computacionais distribuídos podem ser compostos por um conjunto de processos executados por máquinas distintas que coordenam suas operações através da troca de mensagens sobre uma rede de comunicação.

Grande parte das organizações atuais são altamente dependentes de serviços providos de forma distribuída (Hauser et al., 2013). Contudo, estes sistemas possuem vulnerabilidades de segurança que podem ser exploradas intencionalmente ou mesmo acionadas inadvertidamente (Bhargava e Lilien, 2004). Uma forma de enfrentar os problemas originados por estas vulnerabilidades é a implementação de sistemas que disponibilizam mecanismos de proteção. Geralmente, os mecanismos aplicados para esta finalidade atuam como forma de prevenção, tais como políticas de segurança e sistemas de monitoramento.

Neste sentido, a detecção de intrusão é uma atividade amplamente empregada em sistemas computacionais de modo geral e, inclusive, em sistemas distribuídos. Este tipo de tarefa é realizada a partir do monitoramento das redes e de seus componentes (como computadores e aplicações), com o propósito de reconhecer comportamentos de natureza duvidosa ou indícios de possíveis incidentes (Scarfone e Mell, 2012). Frequentemente, a automatização desse processo é efetuada por um sistema de detecção de intrusão (*Intrusion Detection System* - IDS), o qual pode ser constituído por softwares e/ou hardwares implantados em diversos pontos.

A principal responsabilidade de um IDS é realizar a distinção entre comportamentos benignos e maliciosos. Assim, estes sistemas são baseados no reconhecimento de padrões. As abordagens de reconhecimento mais adotadas pelos IDSs são: detecção por assinatura, que se baseia na identificação das estratégias padrões das ameaças; e detecção baseada em anomalias, na qual os comportamentos normais do sistema monitorado são utilizados como base para detecção. A detecção por assinatura é considerada mais simples e efetiva para detectar ataques conhecidos. Por outro lado, a detecção baseada em anomalias possibilita a identificação de potenciais ameaças sem a necessidade de conhecimento prévio dos padrões de ataques, o que é considerado uma vantagem sobre a detecção por assinatura em relação à flexibilidade de detecção de novos ataques (Liao et al., 2013).

Na abordagem de detecção baseada em anomalias, são construídos modelos a partir da observação de comportamentos legítimos ou esperados dos componentes do sistema (por exemplo, usuários, conexões de rede ou aplicativos). Assim, um modelo comportamental é utilizado como referência durante o monitoramento do sistema, com o objetivo de possibilitar a identificação de padrões desconhecidos.

1.1 MOTIVAÇÃO

Apesar de sua ampla adoção e dos diversos benefícios proporcionados, os IDSs tendem a apresentar imprecisões ao detectar intrusões. Especialmente no caso de IDSs baseados em anomalia, a principal causa para as imprecisões é o alto nível de complexidade para se modelar o comportamento normal do sistema a ser monitorado. A seguir serão descritos os principais desafios relacionados à modelagem de sistemas, em particular, os distribuídos.

As tarefas realizadas por um sistema computacional exigem a execução de um conjunto de ações. Neste contexto, cada ação individual é considerada um *evento* (Coulouris et al., 2013).

De modo geral, a detecção de intrusão é baseada na análise de sequências de eventos de um sistema, ordenados através de um relógio global (Totel et al., 2016). Entretanto, em um sistema distribuído real a coordenação entre os processos é realizada sem a presença de um relógio global. Portanto, ordenar globalmente os eventos que ocorreram no sistema é o primeiro desafio a ser superado.

Devido às dificuldades impostas pela ausência de um relógio global, uma característica comum em implantações de IDSs em ambientes distribuídos reais é a utilização da arquitetura baseada em *host*. Nesta abordagem, cada nodo do sistema distribuído conta com uma aplicação de detecção de intrusão individual. Assim, o objetivo permanece o mesmo dos ambientes não distribuídos: detectar ataques isoladamente. A perspectiva de distribuição somente é considerada através de um processo de correlação de alertas (Totel et al., 2016), onde uma entidade superior recebe como entrada todos os alertas gerados por cada um dos IDSs e, só então, avalia o comportamento do sistema como um todo para identificar a ocorrência de um ataque global. Neste caso, a detecção de intrusões é feita sem considerar a relação entre os eventos globais do sistema. Entretanto, ataques podem passar despercebidos, pois determinados eventos só podem ser considerados anômalos quando analisados através de uma visão global do sistema, já que os mesmos podem ser tratados como normais pelos *hosts* individuais.

Outro desafio frequente da modelagem comportamental de sistemas está relacionado ao volume de dados que deve ser manipulado. Isto porque, ao monitorar até mesmo pequenos sistemas de software é produzida uma enorme quantidade de dados (Lorenzoli et al., 2006). Portanto, lidar com os dados coletados de modo a gerar informações suficientes para identificar os comportamentos normais de um sistema é algo realmente desafiador. Neste sentido, um dos principais desafios é armazenar e interpretar adequadamente os dados.

Por fim, como consequência das dificuldades de modelagem dos sistemas, um desafio recorrente ao se projetar IDSs baseados em anomalias é encontrar meios para superar as altas taxas de alertas incorretos. Muitos destes alertas são falsos positivos, ou seja, atividades benignas que são identificadas como maliciosas. Isso prejudica a percepção do administrador sobre as atividades que requerem a devida atenção: as verdadeiras ameaças. Naturalmente, a modelagem comportamental do sistema exerce influência direta nestas altas taxas, pois quanto mais comportamentos normais forem representados corretamente pelo modelo, menor será a quantidade de alertas incorretos gerados pelo IDS.

1.2 PROPOSTA

Este trabalho propõe uma nova abordagem para modelagem comportamental de sistemas distribuídos, visando a detecção de intrusão. O principal objetivo é viabilizar a construção de modelos de comportamento normal de sistemas que possibilitem a detecção de anomalias em aplicações distribuídas reais, visando superar os desafios apresentados na seção anterior.

Um primeiro ponto da presente proposta é considerar a relação entre os eventos globais do sistema. Mais especificamente, a modelagem é realizada a partir de registros (*logs*) de eventos de cada nodo individual do sistema, mas os comportamentos normais são identificados através de sequências de eventos globalmente ordenados. Isso é possibilitado a partir da utilização de relógios lógicos (atribuição de *timestamps*), que aplicando a relação *happened-before* permite construir uma ordenação global respeitando a ordem de causalidade dos eventos (Lamport, 1978).

A representação dos comportamentos normais do sistema pode ser feita utilizando diversas abordagens, como, por exemplo, autômatos (Lanoë et al., 2019), propriedades temporais (Beschastnikh et al., 2011) e modelos difusos (Raguenet e Maziero, 2008). O presente trabalho explora o uso de n-gramas para a construção de modelos de comportamento normal de um

sistema distribuído. Um *n*-grama é uma sequência de *n* eventos envolvendo um ou mais nodos do sistema distribuído. Essa abordagem tem demonstrado resultados promissores em diversas áreas (Sidorov et al., 2014; Santos et al., 2009; Barrón-Cedeño e Rosso, 2009), mas ainda é pouco explorada no contexto da modelagem comportamental de aplicações distribuídas.

1.3 ESTRUTURA DA PROPOSTA

O restante do trabalho está organizado da seguinte forma. O Capítulo 2 contém a fundamentação teórica onde são apresentados conceitos envolvendo sistemas distribuídos, detecção de intrusão e técnicas para modelagem de comportamento normal de sistemas. No Capítulo 3 são apresentados alguns dos trabalhos que têm sido realizados no contexto desta pesquisa. A proposta da dissertação está contida no Capítulo 4. O Capítulo 5 apresenta a avaliação experimental da solução proposta. Por fim, o Capítulo 6 conclui o trabalho e discute os possíveis trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são apresentados os conceitos relacionados ao tema deste trabalho. A Seção 2.1 apresenta os principais fundamentos de sistemas distribuídos. Na Seção 2.2 são descritos os conceitos essenciais de detecção de intrusão. Em seguida, são apresentadas as técnicas mais relevantes para modelagem de comportamento normal de sistemas na Seção 2.3. Por último, a Seção 2.4 conclui o capítulo.

2.1 SISTEMAS DISTRIBUÍDOS

De modo geral, um sistema distribuído é uma coleção de entidades independentes que cooperam com o objetivo de resolver um problema que não pode ser resolvido individualmente (Kshemkalyani e Singhal, 2011). Especificamente no contexto deste trabalho, um sistema distribuído é composto por um conjunto de processos que coordenam suas operações através da troca de mensagens. Um processo é um programa que pode ser executado por qualquer dispositivo de hardware (comumente chamado de nodo em uma rede), como um computador, um celular, entre outros. As trocas de mensagens entre os processos são realizadas através de uma rede de comunicação.

Sistemas distribuídos possuem diversas características importantes. Uma delas é o compartilhamento de recursos. Isso significa que todos os processos que compõem o sistema, compartilham entre si recursos que variam desde entidades lógicas (como arquivos de dados), até dispositivos físicos (como impressoras, discos de armazenamento, dentre outros). Estes recursos podem estar alocados, por exemplo, em uma mesma sala ou ainda em diferentes continentes.

Outra característica relevante é a transparência oferecida por esses sistemas, ou seja, a ocultação da separação dos componentes do sistema em várias partes. O mínimo de transparência que pode ser oferecida é um cenário em que o sistema é visto por seus usuários como um sistema único, não como um conjunto de partes. Porém, é possível que, em um nível elevado de transparência, os usuários sequer consigam identificar se o sistema é centralizado ou distribuído. Existem diversos tipos de transparência, sendo algumas delas a transparência de acesso, de localização e de replicação (Tanenbaum e Van Steen, 2007).

A escalabilidade também é um importante aspecto característico dos sistemas distribuídos. Um sistema é considerado escalável se permanece eficiente quando ocorre uma significativa elevação na quantidade de recursos e usuários (Coulouris et al., 2013). Para viabilizar a escalabilidade, é necessário que gargalos no sistema sejam identificados e tratados adequadamente.

Os sistemas computacionais são sujeitos a falhas, isto é, podem ocorrer casos em que as aplicações não executem suas tarefas conforme a sua especificação. As falhas podem gerar comportamentos inconsistentes e, possivelmente, gerar resultados incorretos. Por isso, outra característica relevante dos sistemas distribuídos é a tolerância a falhas, ou seja, a capacidade do sistema de mascarar falhas que ocorrem em alguns de seus componentes. Vários mecanismos podem ser utilizados para fornecer tolerância a falhas, tais como resiliência do processo, comunicação confiável, confirmação distribuída, acordo e consenso, detecção de falhas e auto-estabilização (Kshemkalyani e Singhal, 2011).

2.1.1 Eventos em Sistemas Distribuídos

A definição de evento em um sistema distribuído pode variar de acordo com o contexto em que o sistema é observado. De forma abrangente, um evento pode ser definido como cada uma das ações individuais realizadas por um processo ao longo de sua execução (Coulouris et al., 2013). Neste trabalho, os eventos estão relacionados às trocas de mensagens entre os processos, portanto, um evento representa uma ação de envio ou recebimento de uma mensagem.

Os eventos ocorridos em um sistema podem ser registrados e armazenados em memória secundária individualmente pelos processos. Cada arquivo de registro de eventos é chamado de *log* ou *traço*. Quando uma aplicação é executada uma quantidade de vezes $(1, 2, \ldots, i)$ para cada execução um novo traço contendo estes eventos é gerado (*i.e.*: $t_1, t_2, \ldots, t_i$). Este tipo de registro é uma prática que tem sido amplamente adotada pelos sistemas computacionais (Du et al., 2017). Os *logs* são fontes de informações simplificadas que podem ser utilizadas, por exemplo, para atividades de diagnóstico de falhas ou análise de segurança dos sistemas (Jia et al., 2017).

2.1.2 Ordenação de Eventos

Cada processo de um sistema distribuído realiza diversos envios e recebimentos de mensagens ao desempenhar suas atribuições. Logo, em cada processo ocorrem vários eventos. Naturalmente, os eventos acontecem em uma determinada sequência $[a,\ b,\ c,\ \ldots]$. Portanto, a sequência em que os eventos ocorrem em cada processo individual é chamada de ordem local.

Muitas vezes é importante identificar em qual ordem os eventos de um sistema ocorreram, isto é, definir uma ordem global da ocorrência de eventos em todo o sistema. No entanto, obter esta ordenação não é uma tarefa trivial, especialmente quando se trata de sistemas distribuídos reais. Isso porque, no contexto atual, é impossível sincronizar com perfeição os relógios locais de todos os nodos que compõem o sistema. Dessa forma, a tentativa de utilização de um relógio físico global pode resultar em uma ordenação incorreta de eventos, principalmente quando os mesmos acontecem com uma diferença de tempo extremamente curta.

Todavia, existem técnicas que, quando aplicadas corretamente, permitem a ordenação global dos eventos sem considerar o sincronismo dos relógios entre os nodos do sistema. Mas para entender como isso pode ser realizado, é interessante primeiramente analisar como é feita a ordenação dos eventos pertencentes a um único processo, ou seja, os eventos locais. Para isso, é possível utilizar a relação "aconteceu antes de" (ou "*happened before*"), denotada como $\rightarrow$ (Lamport, 1978).

A relação "*happened before*", no conjunto de eventos de um sistema, é a menor relação que satisfaz as seguintes condições:

(I) Se a e b são eventos que ocorrem em um mesmo processo, usando o relógio local é possível inferir que a aconteceu antes de b, logo $a \rightarrow b$;

(II) Se $send(m)$ corresponde ao envio de uma mensagem m, e $receive(m)$ é a recepção desta mensagem em outro processo, então $send(m) \rightarrow receive(m)$;

(III) Se $a \rightarrow b$ e $b \rightarrow c$, então $a \rightarrow c$.

Através da relação $\rightarrow$ é possível definir uma ordem parcial dos eventos do sistema, porém ela não é suficiente para encontrar a ordem total (global), pois existem eventos concorrentes em que não se pode afirmar que $a \rightarrow b$ ou $b \rightarrow a$. Isto pode ser representado, respectivamente, como $a \nrightarrow b$ e $b \nrightarrow a$. Eventos concorrentes acontecem na presença de eventos distintos de diferentes processos em que não há encadeamento de mensagens entre eles, sendo tais eventos

denotados como $a||b$. Entretanto, há uma extensão desta abordagem, chamada *relógios lógicos*, que introduz a relação $\Rightarrow$ e permite alcançar uma ordenação total válida dos eventos (Lamport, 1978).

O conceito de relógio lógico é baseado na existência de contadores locais (internamente a cada processo) que permitem atribuir *timestamps* (ou carimbos de tempo) aos eventos que ocorreram em diferentes processos do sistema distribuído. Este contador local é incrementado monotonicamente e não possui nenhuma ligação em particular com qualquer relógio físico. Este incremento é feito no contador de cada processo p_i através de uma função C_i, presente em todos os processos, que atribui um valor inteiro $C_i(a)$ para todo evento a que ocorreu no processo p_i. Para capturar as relações entre os eventos, os processos atualizam seus respectivos relógios lógicos e transmitem seus valores nas mensagens trocadas entre si, de acordo com as seguintes regras:

(I) O processo p_i incrementa seu contador local ($C_i = C_i + 1$);

(II) O processo p_i envia uma mensagem m acompanhada de um *timestamp* t, indicando o valor de seu contador local C_i ($send(m, t)$);

(III) O processo p_j recebe a mensagem ($receive(m, t)$), atualiza o valor de seu contador local aplicando a regra $C_j = max(C_j, t)$, aplica a regra (I) e assinala o resultado como indicativo de tempo para o recebimento da mensagem.

2.1.3 Cortes e Séries em *Logs* de Eventos

Conforme mencionado na Subseção 2.1.1, cada processo individual pode registrar seus eventos em arquivos de *logs*. Ao fazer isso, cada processo registra seus eventos em uma sequência específica de ocorrência, chamada de ordem local (como foi apresentado na Subseção 2.1.2). Logo, um *log* de eventos de um processo individual é considerado o seu histórico local de eventos. A partir da união dos históricos locais de todos os processos, é possível obter o histórico global do sistema.

Com base no histórico global é possível realizar diversas análises de um sistema. Entretanto, devido às características dos sistemas distribuídos, esta não é uma tarefa trivial. Por exemplo, por conta dos atrasos na transmissão de mensagens, quando um processo envia uma mensagem para múltiplos processos, é impossível garantir que todos os processos de destino executem as ações locais simultaneamente. Isso motiva uma noção de particionamento do histórico global em fatias de tempo, através de um procedimento chamado corte (Mattern et al., 1988). Assim, um corte é um subconjunto do histórico global do sistema (Coulouris et al., 2013).

Graficamente, um corte é representado por uma linha em zigue-zague cortando um diagrama de tempo em duas partes (uma parte esquerda e uma parte direita). A Figura 2.1 ilustra dois cortes em um diagrama de tempo envolvendo dois processos. Um corte pode ser consistente ou inconsistente. A linha tracejada na Figura 2.1 representa um corte inconsistente, pois a história de p_2 até o ponto de corte inclui o recebimento da mensagem m_1, mas a história de p_1 até o ponto de corte não inclui o envio desta mensagem (ou seja, um efeito sem causa). Já a linha contínua exemplifica um corte consistente, uma vez que ele inclui tanto o envio quanto a recepção da mensagem m_1. Pode-se observar que o corte representado pela linha contínua inclui o envio de m_2, mas não o seu recebimento em p_1. Mesmo assim, este corte é considerado consistente, pois em um sistema real a mensagem leva certo tempo para chegar ao destino (Coulouris et al., 2013).

Neste sentido, outro conceito relevante é a ordenação total de todos os eventos de um histórico global consistente com cada ordem do histórico local, sendo definido como série. Já

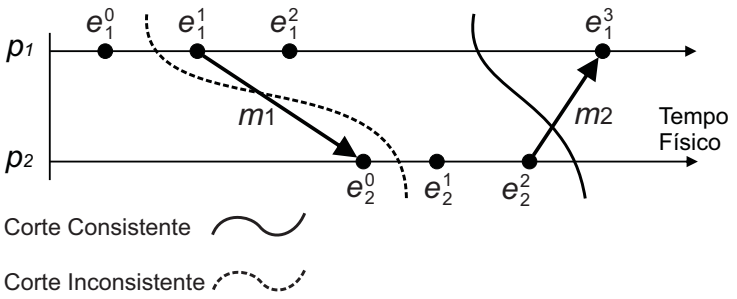


Figura 2.1: Exemplo de corte consistente e corte inconsistente. Fonte: (Coulouris et al., 2013)

uma série consistente é uma ordenação dos eventos em um histórico global consistente com a relação $\rightarrow$ no histórico global (Coulouris et al., 2013).

2.1.4 Trocas de Mensagens entre Processos

A coordenação entre os processos de um sistema distribuído é feita através da troca de mensagens. Para isso, é preciso haver um canal de comunicação que permita que uma mensagem enviada por um processo seja encaminhada e recebida pelo processo (ou processos) de destino. Contudo, é possível que ocorram falhas na transmissão de mensagens. Uma solução para este tipo de problema é o mascaramento de falhas, que viabiliza a comunicação confiável entre os processos do sistema. Um canal que oferece comunicação confiável garante que: toda mensagem enviada é entregue ao seu destino (independentemente do tempo necessário); cada mensagem recebida é idêntica à enviada; e nenhuma mensagem é entregue duas vezes (Coulouris et al., 2013).

Uma consequência do mascaramento de falhas é a possibilidade de ocorrência de reenvio de mensagens. O motivo é que geralmente isso é feito através de um esquema de confirmação que verifica se cada mensagem foi corretamente entregue ao destino. Neste esquema, sempre que o recebimento de uma mensagem não é confirmado, a mensagem é reencaminhada. Nesse caso, considerando que cada processo registra seus eventos no nodo em que executa, enquanto no nodo de destino há somente um único registro de evento de recebimento da mensagem, no nodo que executa o processo remetente pode haver múltiplos registros de envio da mesma mensagem. No contexto deste trabalho, estes registros redundantes são considerados eventos espúrios, ou seja, são desconsiderados na modelagem do sistema.

2.2 DETECÇÃO DE INTRUSÃO

Um sistema computacional é construído para exercer funcionalidades específicas, visando atender as necessidades dos usuários para os quais foi projetado. Durante sua execução, cada sistema apresenta um determinado padrão de funcionamento, principalmente porque existem diversas regras preestabelecidas para a comunicação entre seus componentes e até mesmo com os usuários. Este padrão representa o comportamento normal do sistema.

Entretanto, qualquer sistema pode estar sujeito a sofrer ataques (ou intrusões) efetuados por entidades maliciosas, que por variados motivos, visam obter acesso a recursos do sistema de forma indevida. Quando um invasor executa um ataque, na maioria das vezes, suas ações provocam comportamentos anormais no sistema computacional, pois suas requisições costumam divergir das realizadas habitualmente pelos usuários do sistema.

À medida que se obtém conhecimento de suas vulnerabilidades e, consequentemente, da possibilidade da ocorrência de intrusões, é preciso tomar providências. A identificação de

atividades de intrusão permite desde uma melhor preparação da segurança do sistema para evitar novas invasões, até o desenvolvimento de métodos que possibilitem realizar o bloqueio de intrusões identificadas, caso aconteçam novamente. Desse modo, a detecção de intrusão é o processo de monitoramento das redes e de seus componentes, como computadores ou aplicações, com o propósito de reconhecer atividades de natureza duvidosa ou indícios de possíveis incidentes (Scarfone e Mell, 2012).

Efetuar a detecção de intrusões manualmente, além de ser uma tarefa árdua para os administradores de sistemas (encarregados pela segurança), normalmente não é muito eficaz e exige um alto custo operacional para ser cumprida com sucesso. No intuito de oferecer melhores resultados ao processo de detecção, surgem os sistemas de detecção de intrusão (*Intrusion Detection System* - IDS). Um IDS efetua o monitoramento das atividades do sistema, analisando-as para reconhecer possíveis incidentes, que são ameaças iminentes ou violações reais das políticas de segurança definidas (Angiulli et al., 2015).

Assim, a principal função de um IDS é identificar prováveis incidentes e relatar aos administradores do sistema para que, caso alguma vulnerabilidade seja explorada com sucesso, ações possam ser tomadas para reduzir os danos ocasionados ou até mesmo evitá-los. Registrar informações sobre eventos observados, indicar violações ou problemas relacionados às políticas de segurança e produzir relatórios, são outras funções que um IDS pode exercer.

Mesmo com uma ampla adoção da tecnologia IDS, ainda existem aspectos a serem melhorados e diversos desafios a serem superados. Um IDS não é capaz de realizar detecções de forma perfeitamente segura em um sistema; isso acarreta, em muitos casos, na apresentação de um alto número de alarmes incorretos aos administradores. Estes alarmes incorretos podem ser divididos em falsos positivos (quando um comportamento normal é considerado um ataque) ou falsos negativos (quando ataques são considerados atividades normais) que não geram avisos e devem ser detectados através de análise manual. Consequentemente, isso prejudica a percepção de quais atividades realmente são ameaças em potencial, dificultando a tomada de ações de bloqueio de comportamentos maliciosos. Por isso, a redução da quantidade de falsos positivos e falsos negativos é desejável em um IDS.

2.2.1 Métodos de detecção de intrusão

Um IDS precisa seguir uma estratégia para conseguir identificar efetivamente as ações maliciosas e prejudiciais que ocorrem no sistema. Com esta finalidade, uma ou mais técnicas de detecção de intrusão podem ser aplicadas a um IDS e, dependendo da opção escolhida, sua forma de reconhecer ataques muda. As técnicas de detecção mais comuns são: detecção por assinatura e detecção por anomalia.

A detecção de intrusão baseada em assinatura (*Signature-based Detection* - SD) é considerada a forma mais simples de detecção. Ao usar esta abordagem, o IDS precisa conhecer previamente o perfil do ataque, onde cada atividade maliciosa é descrita por uma assinatura padrão usando sua própria linguagem descritiva (Catania e Garino, 2012). O comportamento malicioso é detectado quando o estado do sistema monitorado corresponde ao mesmo padrão de ataque previamente observado. Este reconhecimento é feito a partir de uma base que contém os traços de vários ataques, assim, esses traços formam as assinaturas das intrusões (Scarfone e Mell, 2012). Logo, somente a partir do momento em que o IDS aprende como funciona um ataque, ele é capaz de identificá-lo em caso de uma nova ocorrência.

Um bom motivo para usar detecção por assinatura é que ela apresenta uma taxa de falsos positivos mais baixa que na abordagem baseada em anomalias (Garcia-Teodoro et al., 2009). A principal desvantagem deste tipo de detecção é que dificilmente a base utilizada contemplará todos os tipos de ataques possíveis que o sistema pode sofrer. Principalmente porque novas

vulnerabilidades podem surgir a todo momento, e uma nova possibilidade de ataque, até então desconhecida, não será adicionada à base. Outra peculiaridade desse tipo de detecção é que grande parte das assinaturas são bastante específicas, assim ataques variantes não são detectados.

A abordagem de detecção por anomalia (*Anomaly-based Detection* - AD) utiliza perfis de comportamentos normais do sistema. Para isso, faz-se necessário o uso de uma base de traços do sistema para que tais comportamentos normais sejam identificados. Portanto, os IDSs baseados na detecção de anomalias analisam primeiramente os dados e constroem padrões de comportamento normal do sistema na fase de treinamento. Mais tarde, quando estiverem monitorando o sistema, comparam os dados recebidos com os padrões de comportamento normal previamente identificados (Zolotukhin e Hämäläinen, 2013). Assim, qualquer desvio destes modelos é considerado um possível ataque.

A grande vantagem da detecção baseada em anomalia é que ela permite a descoberta de novos tipos de ataques sem nenhum conhecimento prévio do novo comportamento malicioso descoberto (Scarfone e Mell, 2012). Por outro lado, devido ao fato de que nem todas as atividades anômalas são realmente maliciosas, IDSs desse tipo tendem a gerar um número consideravelmente maior de alarmes falsos que os sistemas baseados em assinaturas, ou seja, a quantidade de falsos positivos é maior.

Há ainda a possibilidade de se elaborar IDSs híbridos, que utilizam mais de uma técnica, visando fornecer uma detecção mais extensa e precisa. Um bom exemplo é o uso da técnica de detecção baseada em assinatura juntamente a detecção por anomalia, pois elas podem ser consideradas complementares, já que a primeira diz respeito a certos ataques/ameaças e a última se concentra em ataques desconhecidos (Liao et al., 2013).

2.2.2 Métodos de implementação e Arquiteturas

Basicamente, IDSs podem ser implementados de duas formas: baseados em *host* (*Host-Based Intrusion Detection System* - HIDS) ou baseados em rede (ou *Network-Based Intrusion Detection System* - NIDS) (Nisioti et al., 2018). Um HIDS é capaz de identificar atividades maliciosas apenas do *host* em que é implementado. Para isso, monitora os eventos em busca de atividades que vão contra as políticas ou padrão de uso do sistema. Alguns exemplos de fontes de monitoramento são *logs* do sistema local, chamadas do sistema, processos em execução, acesso e modificação de arquivos, entre outros. Este tipo de IDS é comumente implantado em elementos críticos, como servidores que são acessíveis ao público e aqueles que contêm informações confidenciais (Scarfone e Mell, 2012).

Já um NIDS monitora e analisa o tráfego de rede buscando identificar comportamentos suspeitos. Para tal, é realizada uma inspeção detalhada do conteúdo dos pacotes (*Deep Packet Inspection* - DPI), que verifica o cabeçalho e a carga útil de cada pacote (Nisioti et al., 2018). Este método é capaz de monitorar as atividades de múltiplos nodos em simultâneo. Uma das suas principais vantagens, se comparado com um HIDS, é a independência de plataforma, podendo ser implementado em diferentes plataformas sem a necessidade de modificação. Devido à alta complexidade das redes atuais, o monitoramento pode ser realizado de diferentes formas. Por exemplo, monitorar somente alguns protocolos ou aplicações individuais da rede, ou baseado em rede sem fio, sendo projetado especialmente para monitorar o tráfego (ou protocolos específicos) destas redes.

A arquitetura do sistema de detecção também é um fator importante para o seu desempenho. Um IDS centralizado utiliza vários sensores de monitoramento em diferentes pontos da rede, esses sensores enviam informações para uma unidade central de processamento, que é a responsável por correlacionar esses dados e detectar comportamentos suspeitos. Já em um IDS distribuído, esta unidade de processamento é implementada em locais diferentes,

se tornando agentes autônomos responsáveis pelo monitoramento e processamento. A carga de trabalho é distribuída entre estes agentes, que se comunicam e compartilham informações. Existe também o IDS descentralizado (Nisioti et al., 2018). Neste cenário, diversos sensores e unidades de processamento são implementados na rede, organizados hierarquicamente. Os dados de monitoramento são encaminhados e pré-processados pelos agentes mais próximos, antes de serem analisados pela unidade de processamento principal.

2.3 MODELAGEM DE COMPORTAMENTO NORMAL DE SISTEMAS

Conforme mencionado na Seção 2.2.1, os IDSs baseados em anomalia necessitam de um modelo contendo os comportamentos normais e corretos do sistema que será monitorado. Este modelo será utilizado como referência durante o monitoramento, onde qualquer comportamento do sistema que não estiver condizente com o modelo será apontado como uma anomalia.

Um modelo comportamental é um modo de representação do sistema monitorado. Ele deve conter características suficientes para distinguir um comportamento comum de uma anomalia. Um comportamento do sistema nada mais é do que a sequência de estados de cada nodo ou processo durante uma execução. Dessa forma, os sistemas podem apresentar infinitos comportamentos.

Geralmente, os modelos são construídos a partir da observação do sistema durante um intervalo de tempo. Durante essa fase é criada uma base contendo informações específicas sobre os acontecimentos do sistema. Essas informações variam de formato conforme o sistema a ser modelado, podendo ser arquivos de *logs*, pacotes de rede, entre outros. Posteriormente, essa base é trabalhada para que informações irrelevantes sejam eliminadas, resultando em um modelo primário.

2.3.1 Técnicas para Modelagem de Comportamento Normal

Existem várias técnicas que podem ser empregadas para a construção do modelo do sistema. De acordo com (Garcia-Teodoro et al., 2009) podemos classificá-las em três principais categorias: baseada em estatística, baseada em conhecimento e baseada em aprendizado de máquina.

A técnica baseada em estatística utiliza perfis estocásticos do sistema para realizar a detecção. Durante o monitoramento, uma pontuação de anomalia é estimada para o comportamento atual em relação ao modelo. Esta pontuação determina um grau de diferença para o comportamento, o sinalizando como uma anomalia caso ultrapasse um limite. Dentro deste contexto existem as séries temporais, que consideram a ordem e o contador de eventos entre as ocorrências, bem como seus valores. O tráfego monitorado é considerado anormal caso a probabilidade de retorno à sua normalidade for muito baixa a partir de um determinado momento (Khraisat et al., 2019).

Os sistemas baseados em conhecimento modelam o sistema utilizando algum tipo de ferramenta formal, para que seja aplicado um conjunto de regras que possibilite a detecção de anomalias. Estes conjuntos de regras determinam os comportamentos corretos que o sistema pode assumir. O modelo é construído manualmente por um especialista humano, a partir de dados extraídos do ambiente. Os dados são analisados e modelados seguindo a ferramenta formal. Uma ferramenta formal comumente utilizada são as máquinas de estado finito.

Máquinas de estado finito (*Finite-State Automaton* - FSA), também chamados de autômatos, são estruturas que contêm um número finito de estados e produzem saídas nas transições de estados após o recebimento de entradas (Lee e Yannakakis, 1996). Elas são

utilizadas para modelar sistemas em diversas áreas, pois facilitam a compreensão sobre possíveis eventos que podem acontecer a partir do fornecimento de um conjunto de entradas e observação das respectivas saídas.

Outra ferramenta formal é a linguagem de descrição, que define a sintaxe das regras que podem ser usadas para especificar as características de um comportamento (Khraisat et al., 2019). Uma opção é o método de n-gramas, onde um n-grama consiste em um fragmento de n itens consecutivos (letras, palavras, mensagens, eventos, etc.), de uma sequência de eventos do sistema (Zolotukhin e Hämäläinen, 2013). Formalmente, dada uma *string* X que contém uma sequência de símbolos pertencente a um alfabeto A, obtém-se o conjunto de n-gramas de X lendo-se X sequencialmente com uma janela de comprimento n. O tamanho da janela pode variar de acordo com a estratégia utilizada pelo autor ou tipo de dado. Para ilustrar o conceito de n-grama, pode-se considerar uma sequência de eventos $[a, b, c, d, e, f, g]$. Assumindo $n = 3$, os 3-gramas presentes nessa sequência são: abc, bcd, cde, def e efg.

A técnica de aprendizado de máquina consiste em extrair conhecimento de grandes quantidades de dados (Khraisat et al., 2019). Para isso, se faz necessário possuir dados rotulados para treinar o modelo comportamental. A partir desses dados são identificados padrões que são utilizados para reconhecer ou até mesmo prever um comportamento. Vários métodos podem ser utilizados, como redes neurais, agrupamento, árvores de decisão, entre outros. A estratégia mais comum nesse contexto é utilizar duas bases de dados na fase de treinamento, uma contendo os comportamentos corretos do ambiente e outra com assinaturas de ataques, de forma que o comportamento seja classificado como normal ou anômalo.

2.4 CONSIDERAÇÕES FINAIS

No contexto deste trabalho, um sistema distribuído é tratado como um conjunto de processos que coordenam suas operações através da troca de mensagens. Um processo é entendido como um programa que pode ser executado por qualquer dispositivo de hardware. Um ponto central dos sistemas distribuídos é o compartilhamento de recursos, físicos e lógicos, que são disponibilizados a todos os processos que compõem o sistema. Tais processos executam um conjunto de ações para a realização de tarefas, e cada ação individual é denominada como *evento*, com vários eventos acontecendo em cada processo. Por fim, a coordenação de tarefas em sistemas distribuídos geralmente depende de uma noção compartilhada de tempo. Devido às características de arquitetura e comunicação, em sistemas distribuídos não há uma noção única e global de tempo, sendo necessária a utilização de algoritmos específicos para determinar a ordenação de eventos.

A ordenação de eventos ocorridos em um único processo é facilmente obtida através da utilização do relógio local, mas, quando se trata de sistemas distribuídos, essa ordenação não é trivial. Como não é possível alcançar uma sincronia entre todos os relógios de um sistema distribuído, se torna inviável utilizar o tempo físico para obter a ordenação dos eventos globais. Desta forma, técnicas que não são baseadas em tempo físico podem ser aplicadas para obter uma ordenação parcial dos eventos, tais como a "aconteceu antes de" (ou "*happened before*").

Comumente, cada processo registra e armazena seus eventos em arquivos de *logs*. Sendo assim, é possível obter um histórico global do sistema a partir da união dos históricos locais de cada processo. Para que uma análise correta seja realizada sobre esses *logs*, o corte precisa ser consistente.

Sistemas de detecção de intrusão (IDSs) são responsáveis por efetuar o processo de detecção. Estes podem ser baseados em assinaturas, que utilizam uma base de dados com perfis de comportamentos reconhecidos como ataques. Embora seja eficiente para detectar

ataques conhecidos, quaisquer variantes dos mesmos ataques não são detectadas. Outra possível abordagem é a detecção por anomalia, que utiliza perfis de comportamentos normais do sistema como base para detecção. Sendo assim, nenhum conhecimento prévio sobre possíveis ataques é necessário.

Durante o período de aprendizagem, um IDS baseado em anomalia realiza a análise dos dados e a construção de um modelo de comportamento normal. Existem três técnicas que podem ser empregadas para este fim: baseada em estatística, baseada em conhecimento e baseada em aprendizado de máquina. Depois, durante a fase de detecção, o comportamento atual do sistema é confrontado ao modelo construído, em busca de comportamentos anômalos. Assim, qualquer desvio desse modelo é considerado um ataque.

3 TRABALHOS RELACIONADOS

Este capítulo apresenta os trabalhos de maior relevância que propõem soluções para construção de modelos de normalidade de sistemas computacionais em contextos similares à proposta do presente trabalho. Na Seção 3.1 são apresentadas algumas das principais abordagens para construção de modelos de normalidade de sistemas distribuídos, com base em conhecimento do sistema. A Seção 3.2 discorre sobre os principais trabalhos que empregam *n*-gramas em soluções para detecções de anomalias. Por fim, a Seção 3.3 discute os trabalhos apresentados, estabelecendo suas relações, bem como evidenciando suas influências no presente trabalho.

3.1 MODELAGEM DE SISTEMAS DISTRIBUÍDOS

Autômato é uma ferramenta formal que tem sido amplamente utilizada em soluções de modelagem de sistemas distribuídos baseadas em conhecimento. Geralmente, o emprego de autômatos é feito em conjunto com alguma técnica temporal. A seguir, são apresentados alguns dos trabalhos mais relevantes e recentes que realizam este tipo de modelagem.

Visando oferecer maior qualidade de serviço através da detecção de anomalias em um sistema distribuído, (Fu et al., 2009) propõem uma ferramenta para realizar o monitoramento dos *logs* e efetuar a identificação de anomalias. Primeiramente, é apresentado um algoritmo de filtragem em *logs*, que converte mensagens de texto em chaves de *logs*. Com isso, é realizada a leitura de informações não estruturadas que são frequentemente encontradas em um ambiente distribuído e a extração dos dados relevantes. A partir destas informações são gerados dois modelos principais: um autômato que representa o fluxo de trabalho normal para cada componente do sistema e um modelo gaussiano (Rasmussen, 2014) que verifica o tempo de execução entre as transições de estado. Os resultados apontam que a proposta é capaz de detectar anomalias relacionadas a largura de banda reduzida e também de atraso entre transições de estado.

A proposta apresentada por (Totel et al., 2016) visa combinar dois formatos de modelos que costumam ser usados separadamente: autômatos e propriedades temporais. O autômato representa os possíveis estados que o sistema pode assumir, já as propriedades temporais estabelecem a sequência em que os estados ocorrem. Em ambos os formatos, os modelos são gerados a partir de traços de eventos capturados de várias execuções finitas de uma aplicação distribuída, sendo posteriormente ordenados globalmente com base em (Lamport, 1978). Para se construir um único autômato global, é inicialmente criado um reticulado (*lattice*) que identifica novas sequências que não foram notadas com a ordenação dos eventos, em seguida todas as sequências identificadas são mescladas e generalizadas. As propriedades temporais são geradas com base no tipo e ordem de ocorrência dos eventos, sendo cada propriedade válida para todas as execuções registradas pelo autômato. Os resultados demonstram que na medida em que se aumenta o número de traços para criação do autômato, mais estados são aprendidos, porém, menos propriedades temporais são atendidas. Além disso, quanto maior quantidade de traços, menor é a taxa de falsos positivos.

O trabalho (Lanoë et al., 2019) é um aprimoramento da abordagem utilizada por (Totel et al., 2016), com o intuito de aumentar a eficiência na fase de construção do autômato. Para isso, duas soluções são propostas, a primeira visa diminuir o tamanho dos autômatos intermediários, onde ao invés de utilizar toda a estrutura de treliça gerada, somente as sequências mais significativas são selecionadas. A segunda proposta investiga o tempo gasto na fase de generalização, para isso durante a fase de construção do autômato são gerados vários modelos

intermediários que são repetidamente mesclados e generalizados até obter seu estado final. Os resultados demonstraram que realizar várias chamadas ao módulo de generalização é mais vantajoso que somente uma única chamada, diminuindo em cerca de 5 vezes o tempo necessário, afetando de forma insignificante as taxas de falso negativo. Entretanto, em termos de falsos positivos, ambas abordagens se mostraram similares.

Um estudo das relações causais entre serviços é apresentado em (Jia et al., 2017), onde é proposto um modelo de grafo híbrido que captura fluxos de execuções normais entre serviços e também internos a um serviço. Sendo assim, a primeira camada do grafo registra a frequência dos *logs* na intenção de representar como diferentes solicitações são encaminhadas entre os nodos. Já a segunda camada é um grafo de controle de fluxo com limite de tempo, que visa demonstrar a duração do processamento completo das solicitações pelo sistema. Com base nessas informações, foi possível categorizar as anomalias para serem analisadas em anomalias de sequência (quando a sequência de fluxo não ocorre dentro da ordem e do tempo esperado), anomalias de latência (quando a sequência de fluxo esperada é obedecida, porém, excede o limite de tempo) e também anomalias de redundância. Os resultados apresentados revelam que a bidirecionalidade nos grafos gerados influencia diretamente na taxa de falsos positivos, aumentando o custo de detecção e diminuindo a acurácia do modelo.

3.2 MODELAGEM DE SISTEMAS BASEADA EM *N*-GRAMAS

Um n-grama consiste em um fragmento de n itens sobrepostos (letras, palavras, mensagens, eventos, etc.), de um certo seguimento (Zolotukhin e Hämäläinen, 2013). Esse conceito tem sido aplicado em diversas áreas de pesquisa, sendo assim, o foco desta seção é apresentar a utilização de n-gramas em soluções para detecções de anomalias.

Um dos primeiros trabalhos a utilizar n-gramas para realizar detecção de anomalias é (Forrest et al., 1996), onde n-gramas de sequências de chamadas de sistemas realizadas por processos Unix privilegiados são empregadas para definir o comportamento normal do sistema. Uma base de dados de padrões normais do sistema é construída, através do uso de uma janela deslizante de tamanho $i + 1$, nos traços obtidos dos processos em execução. Nesse caso, i se refere a sequência de chamadas de sistemas presentes no traço, selecionadas para compor a grama. Um comportamento anormal é detectado quando as sequências do novo traço observado são incompatíveis com as sequências presentes na base. Com isso, o trabalho demonstrou que é possível capturar os padrões de normalidade usando n-gramas curtas (especificamente 5, 6 e 11). Os resultados apresentados por esta pesquisa estimularam novas investigações acerca do uso de n-gramas para definir padrões de comportamentos de sistemas.

O uso de n-gramas é combinado a autômatos em (Jiang et al., 2006), no intuito de realizar detecção de falhas em aplicações Java. As gramas representam as subsequências contíguas de solicitações entre os componentes de um traço, respeitando as restrições de ordem local e global de ocorrência, enquanto os autômatos são utilizados para conectar os n-gramas afim de caracterizar esses traços. Para isso são propostos dois algoritmos, o primeiro extrai as gramas de tamanho variado baseado em um limite, selecionando apenas as gramas mais frequentes nos traços. Já o segundo é responsável pela construção dos autômatos, na intenção de representar uma série de ocorrência entre as gramas calculadas. A solução se mostrou satisfatória na detecção de anomalias de "chamadas nulas" e "exceção esperada".

McPAD (*Multiple Classifier Payload-based Anomaly Detector*) (Perdisci et al., 2009), introduz de um detector baseado em carga útil capaz de identificar código *shell* malicioso presente em pacotes de rede. Para efetuar esta tarefa, são extraídas informações da carga útil através de uma janela deslizante que envolve 2 bytes presentes em v posições diferentes entre si, tratando

isso como 2_v-gramas. O valor de v representa então o número de amostras de comprimento 2 da carga útil. Através da variação de v, a carga útil é parcialmente reproduzida com o foco em diferentes características, que são exploradas por diversos classificadores. Para amenizar a dimensionalidade dessas amostras, um algoritmo de agrupamento é aplicado. Para realizar a classificação, vários SVMs (*Support Vector Machine*) são usados e sua saída é combinada por um meta-classificador em uma previsão de classificação final, onde o voto da maioria é utilizado. O trabalho realizou uma comparação com a abordagem apresentada em (Wang e Stolfo, 2004), conquistando melhores resultados para a detecção de ataques que carregam alguma forma de código executável na carga útil, como código Shell, devido à distribuição dos bytes da carga útil dos pacotes mudar bastante em relação à distribuição normal.

A proposta apresentada em (Wressnegger et al., 2013) busca identificar quais são os ambientes em que a detecção de anomalias é mais favorável que a classificação de gramas em malignas e benignas, ou vice-versa. São elencados três critérios para avaliação dos dados: perturbação, densidade e variabilidade dos dados. O estudo discute vários tipos de dados que podem ser representados em gramas, como textos e binários de protocolos, além de traços de chamadas de sistema, avaliando os critérios citados para cada tipo de dado. A eficácia da detecção e da classificação é apresentada para os dados pertencentes a um ambiente Web, sendo estes divididos entre dados para detecção de intrusão no lado cliente (baseados em códigos JavaScript) e também no lado servidor (baseados em requisições HTTP). É importante salientar que para realizar os experimentos foi utilizado um conjunto de dados maliciosos na fase de treinamento. Os experimentos realizados constatam que dados de JavaScript dinâmicos são mais adequados para classificação, enquanto JavaScrip "estático" são mais precisos quando utilizados em detecção. Já as requisições HTTP são cabíveis em ambos os cenários (classificação ou detecção).

Uma técnica fundamentada na carga útil de pacotes de rede, capaz de detectar ataques baseados em conteúdo efetuados em protocolos da camada de aplicação como HTTP e FTP, é apresentada em (Angiulli et al., 2015), denominada PCkAD (*Packet Chunk Anomaly Detector*). As gramas representam conjuntos de caracteres mais frequentes nos pacotes de rede. Para extrair as gramas, primeiro são retiradas apenas as informações úteis dos pacotes e então as informações são divididas em partes não sobrepostas e de tamanhos iguais, a partir disso é utilizada janela deslizante que irá de fato extrair as gramas, o tamanho dessa janela possui o mesmo tamanho da grama, neste caso são apresentados 2-grama e 5-grama. As conclusões apresentadas na proposta é que para um valor de n baixo o tamanho do conjunto de n-gramas gerados é consideravelmente pequeno, dificultando o reconhecimento de ataques. No entanto, a atribuição de altos valores a n facilita a identificação de uma anomalia e também produz mais falsos positivos.

Seguindo o método de 2_v-gramas, que se trata da extração de informações da carga útil usando uma janela deslizante envolvendo 2 bytes presentes em v posições diferentes entre si, (Nguyen et al., 2016) busca melhorar a representação das informações estruturais da carga útil dos pacotes. Para isso, apresenta uma técnica que contabiliza a frequência de pares de bytes separados por uma distância menor ou igual a v bytes. Basicamente, utiliza v extratores de características, variando a distância de 0 até v, capturando assim a quantidade de vezes que cada par de bytes está presente na carga analisada. Por fim, obtém um novo vetor através da soma das contagens para fornecer um valor às características extraídas. A solução tem como base os conceitos de (Perdisci et al., 2009), consequentemente, classificadores SVMs são treinados e utilizados nas detecções. Os resultados indicam que a técnica é capaz de melhorar, razoavelmente, a detecção de ataques genéricos[1], ou seja, que não possuem em sua carga algum tipo de código executável.

[1]Alguns exemplos são: divulgação de arquivo, erro de validação de entrada, entre outros.

A abordagem publicada em (Maslan et al., 2018) propõe a utilização de *n*-gramas e similaridade de cossenos para detectar códigos maliciosos em pacotes de rede de protocolos da camada de aplicação. Para construir o modelo inicial, é utilizada uma janela deslizante para extrair conjuntos de caracteres que equivalem aos *n*-gramas. Posteriormente, os *n*-gramas selecionados são representados no formato de vetores *n*-dimensionais. A detecção tem como base a fórmula de similaridade de cossenos, que verifica a semelhança entre dois vetores, um derivado da ação que está sendo analisada e outro que está presente no modelo. Os resultados apresentados revelaram que ambos os vetores (o modelo e do tráfego analisado) são semelhantes, dificultando assim a diferenciação de um código malicioso e um benigno, e diferentes cargas compartilham áreas comuns.

Uma abordagem para detecção estatística de *malware* é apresentada em (Vidal e Monge, 2019), tendo como foco principal a detecção de ataques por imitação. A proposta realiza uma combinação de vários métodos no intuito de que um complemente o outro. São construídos dois modelos, um que representa o comportamento normal e outro que contêm traços de *malwares*, através de *Counting Bloom Filters* (CBF) (Shana e Venkatachalam, 2015), que armazenam a frequência de ocorrência de cada *n*-grama presente na carga útil dos pacotes. Na fase de treinamento, ambos os modelos são utilizados pelo PAYL (*Anomalous Payload-Based Intrusion Detection*) (Wang e Stolfo, 2004) para compreender quais as diferenças entre um comportamento normal e um comportamento suspeito e enfim ser capaz de realizar detecções, que consistem na verificação da similaridade dos pacotes com os modelos aprendidos. A proposta também utilizou como base as soluções APAP (Villalba et al., 2017) e APACS (Vidal et al., 2017). Os resultados demonstraram que a solução manteve um desempenho similar aos trabalhos anteriores (utilizados como base). Porém, ao processar cargas úteis maliciosas ocultadas por imitação sua eficácia demonstrou maior vantagem.

3.3 DISCUSSÃO

Os trabalhos relacionados, descritos anteriormente nas Seções 3.1 e 3.2, foram classificados conforme algumas de suas características relevantes, sintetizados na Tabela 3.1. É possível notar que pesquisas que utilizam *n*-gramas como abordagem não consideram a ordenação entre as informações que estão presentes na base. Consequentemente, podemos afirmar que a relação entre as sequências de estados do sistema também é ignorada.

Os trabalhos citados na Seção 3.2 apresentam resultados satisfatórios e demonstram a viabilidade do uso de *n*-gramas na modelagem de sistemas para a detecção de intrusão baseada em anomalias. Entretanto, estes trabalhos não evidenciam se suas soluções são aplicáveis em sistemas distribuídos. Além disso, nestes trabalhos as características da distributividade do sistema não são consideradas (por exemplo, não são tratadas quaisquer questões relacionadas ao sincronismo — ou ausência dele — entre os múltiplos processos do sistema).

A proposta do presente trabalho consiste em explorar esta lacuna da literatura atual, evidenciada na Tabela 3.1. O ponto central da estratégia proposta é aplicar *n*-gramas na construção de modelos para aplicações distribuídas, utilizando *logs* de eventos do sistema como base para a modelagem, o que não é coberto pelos trabalhos correlatos apresentados.

3.4 CONSIDERAÇÕES FINAIS

Autômatos são ferramentas formais frequentemente empregadas para modelar sistemas distribuídos. Geralmente, são adicionadas outras técnicas em conjunto ao autômato para aperfeiçoar o processo de detecção de anomalias. Os trabalhos descritos durante a Seção 3.1

Modelos		**Autômatos**				***n*-gramas**							
Referências		(Fu et al., 2009)	(Totel et al., 2016)	(Lanoë et al., 2019)	(Jia et al., 2017)	(Forrest et al., 1996)	(Jiang et al., 2006)	(Perdisci et al., 2009)	(Wressnegger et al., 2013)	(Angiulli et al., 2015)	(Nguyen et al., 2016)	(Maslan et al., 2018)	(Vidal e Monge, 2019)
Objetivo	Detecção de falhas	✓			✓		✓						
	Detecção de intrusão		✓	✓		✓		✓	✓	✓	✓	✓	✓
Informação na Base	Logs	✓	✓	✓	✓	✓	✓						
	Pacotes de rede							✓	✓	✓	✓	✓	✓
Técnica utilizada para extrair Estados/ Gramas	*K-means*	✓											
	DFS		✓	✓									
	PC-algorithm				✓								
	Mineração de padrões sequenciais						✓						
	Janela Deslizante					✓		✓	✓	✓	✓	✓	
	Filtros Bloom												✓
Método de ordenação	*happened-before relation*		✓	✓									
	Causa-efeito				✓								
	Ordem local					✓							
	Não especifica	✓					✓	✓	✓	✓	✓	✓	✓

Tabela 3.1: Síntese dos trabalhos relacionados

apresentam diferentes abordagens para o uso dessa ferramenta e consequentemente, os resultados alcançados. Nestes trabalhos, os estados dos autômatos são extraídos a partir de *logs* do sistema e combinados a técnicas temporais especificas: sequências de transição entre os estados ou tempo de execução entre as transições.

Outra ferramenta formal são os *n*-gramas, que são fragmentos de *n* itens sequenciais. Nos trabalhos citados na Seção 3.2, são utilizados *n*-gramas formados por chamadas de sistemas, solicitações entre os componentes, bytes da carga útil, entre outros. Devido a disposição dos dados para o cálculo dos *n*-gramas, uma técnica comumente empregada é a janela deslizante, com diferentes tamanhos e distâncias de posicionamento. Por fim, as pesquisas apresentaram resultados positivos em relação a modelagem de sistemas utilizando *n*-gramas como base.

4 CONSTRUÇÃO DE MODELOS BASEADOS EM *N*-GRAMAS

Neste capítulo é apresentada uma proposta para a construção de modelos baseados em *n*-gramas para detecção de anomalias em sistemas distribuídos. Os objetivos almejados pela proposta são apresentados na Seção 4.1. Algumas premissas em relação ao sistema distribuído que será modelado utilizando a técnica são apresentadas na Seção 4.2. A Seção 4.3 descreve algumas características esperadas para os dados a serem utilizados durante o processo de modelagem. A abordagem empregada para o cálculo dos *n*-gramas de eventos é apresentada na Seção 4.4. Na Seção 4.5 é descrita a solução para construção dos modelos. Por fim, a Seção 4.6 conclui o capítulo.

4.1 OBJETIVOS

O objetivo geral deste trabalho é alcançar a construção de modelos de comportamentos normais de aplicações distribuídas reais que possibilitem a elaboração de sistemas eficazes de detecção de intrusão baseados em anomalias. Para alcançar este objetivo principal, diversos outros objetivos específicos foram definidos.

Visando considerar a relação dos eventos globais do sistema a ser modelado, um objetivo inicial é atingir uma ordenação global válida dos eventos sem a dependência de um relógio físico global. Outro objetivo, é elaborar uma técnica que permita representar comportamentos normais do sistema através de *n*-gramas de eventos. A partir dos *n*-gramas calculados, pretende-se construir modelos que representem de forma eficaz os comportamentos normais. Por fim, objetiva-se ainda elaborar e aplicar uma estratégia para avaliar a capacidade de detecção de comportamentos anômalos pelos modelos gerados.

4.2 PREMISSAS DO SISTEMA MODELADO

Neste trabalho, um sistema distribuído é um conjunto de processos que se coordenam através da troca de mensagens via rede. Em cada processo, o envio ou recebimento de uma mensagem é considerado um evento. Além disso, cada processo registra seus eventos no nodo que o executa, respeitando a ordenação local.

Outra premissa relevante é que a troca de mensagens entre os processos do sistema é realizada através de canais de comunicação confiáveis. Isso pode ser viabilizado, por exemplo, através do uso do protocolo TCP (Beschastnikh et al., 2014). Consequentemente, é esperado também a ocorrência de mensagens espúrias.

Como mencionado anteriormente na Seção 2.1, apesar de ser possível ter uma noção temporal em sistemas distribuídos reais, é impossível, por exemplo, garantir um limite máximo (e preciso) de tempo para cada troca de mensagens entre os processos do sistema. Portanto, neste trabalho, não se assume a presença de um relógio físico global.

4.3 BASE PARA MODELAGEM

Buscando facilitar o entendimento da técnica proposta, todos os elementos utilizados nesta pesquisa (tais como execuções, traços, etc.) serão descritos e referenciados através de notações matemáticas. Uma notação permite realizar uma representação formal que facilite o pensamento lógico/matemático e a visualização de ideias (Blostein e Grbavec, 1997). Neste

sentido, a Tabela 4.1 contém as notações utilizadas ao descrever a solução descrita neste trabalho. Estas notações são úteis tanto para a compreensão da técnica de modelagem detalhada neste capítulo, quanto para a avaliação apresentada adiante no Capítulo 5.

Notação	**Descrição**
$[a,\ b,\ \ldots]$	uma sequência de eventos ordenados.
n	tamanho de uma grama.
T	um traço.
TC	um traço correto.
$\mathbb{E}$	todas as execuções obtidas do sistema que estão presentes na base.
E	uma única execução distribuída presente em $\mathbb{E}$.
$\mathbb{C}$	subconjunto de execuções corretas presentes em $\mathbb{E}$.
$\mathbb{G}$	uma sequência de n eventos consecutivos (n-grama).
$\mathbb{G}(E, n)$	todos os n-gramas distintos de tamanho n obtidos de uma execução E.
$\mathbb{M}$	modelo(s).
$\mathbb{M}\cup$	modelo(s) de união.
$\mathbb{M} \cup (n)$	modelo(s) de união com gramas de tamanho n.
$\mathbb{M}\cap$	modelo(s) de interseção simples.
$\mathbb{M} \cap (n)$	modelo(s) de interseção simples com gramas de tamanho n.
p	quantidade de partições de $\mathbb{C}$.
$\mathbb{P}$	uma partição composta por múltiplas E.
$\mathbb{M}'_i$	modelo parcial de união.
$\mathbb{M}'_i(n)$	modelo parcial de união com gramas de tamanho n.
$\mathbb{M}\sqcap$	modelo(s) de interseção ampla.
$\mathbb{M} \sqcap (n)$	modelo(s) de interseção ampla com gramas de tamanho n.
L	uma lista com os últimos n eventos observados, em ordem de ocorrência.

Tabela 4.1: Notações utilizadas neste trabalho.

A técnica de modelagem proposta neste trabalho parte do princípio de que exista uma base de dados, que reúne os registros de eventos ocorridos em uma determinada aplicação distribuída. Esta base de eventos deve ser gerada ao longo de várias execuções da aplicação, para que diferentes tipos de comportamentos sejam capturados. Logo, $\mathbb{E}$ define o conjunto de todas as execuções registradas que estão presentes na base: $\mathbb{E} = \{E_1,\ E_2,\ \ldots\}$. Por sua vez, uma execução $E_i \in \mathbb{E}$ é definida como o conjunto de traços dos processos que a compõem: $E_i = \{T_{i1},\ T_{i2},\ \ldots\}$, onde T_{ij} corresponde ao traço do processo p_j na execução E_i.

Nesta pesquisa, o arquivo de registro de eventos de cada processo é nomeado traço. Dessa forma, um traço T_{ij} é definido pela sequência de eventos que ocorreram no processo p_j durante a execução E_i em ordem cronológica: $T_{ij} = [e_{ij1},\ e_{ij2},\ \ldots]$, onde e_{ijk} é o k-ésimo evento ocorrido no processo p_j durante a execução E_i. De maneira simplificada, pode-se escrever $T = [e_1,\ e_2,\ \ldots]$, subentendendo o processo p_j e a execução E_i quando não forem necessários. Consequentemente, E_i corresponde a uma única execução distribuída do sistema, que contém x traços, sendo x a quantidade de processos envolvidos na execução.

A construção apropriada de um modelo depende da garantia de que sejam utilizados apenas comportamentos corretos do sistema, ou seja, traços livres de ataques, bem como de qualquer tipo de anomalia. Logo, $\mathbb{C}$ define o subconjunto de execuções corretas presentes em $\mathbb{E}$ ($\mathbb{C} \subseteq \mathbb{E}$). O traço correto coletado individualmente por processo, representado por TC_{ij}, deve ser composto por eventos que caracterizam execuções finitas da aplicação a ser modelada. Isto

implica que cada execução E_i deve possuir uma devida terminação (não gerando impasses[1]). Os eventos armazenados não devem estar criptografados e devem conter informações relevantes, como, por exemplo, nodo de origem e de destino.

Outra característica relevante é que a base de eventos deve ser composta por execuções com diferentes tempos de duração. A principal motivação é capturar a maior diversidade possível de comportamentos heterogêneos.

4.4 CÁLCULO DOS *N*-GRAMAS

Na presente técnica de modelagem, os comportamentos corretos do sistema são representados por n-gramas de eventos. Estes n-gramas são calculados a partir de uma base de eventos corretos extraídos do sistema a ser modelado. Sendo assim, um n-grama G é definido como uma sequência de n eventos consecutivos $N = [e_1,\ e_2,\ \ldots\ e_n]$ observados em uma dada execução E. O conjunto de todos os n-gramas distintos de tamanho n observados em uma dada execução E é definido por $\mathbb{G}(E, n)$.

Dadas as características da base de eventos considerada (conforme descrito nas Seções 4.2 e 4.3), um primeiro passo necessário é realizar um ajuste nos traços de eventos $\mathbb{C}$ de modo a tratar os possíveis eventos que correspondem a mensagens espúrias. Isso porque, estas mensagens não estão relacionadas a quaisquer sequências de eventos que fazem parte de um comportamento correto do sistema. Deste modo, é realizada uma varredura em cada TC visando identificar os eventos que se enquadram nas seguintes condições:

- Um evento de envio que não possui um evento de recebimento no traço do processo de destino;
- Um evento de recebimento que não possui um evento de envio correspondente no traço do processo de origem;

Assim, cada evento que possuir ao menos uma das características descritas corresponde a uma mensagem espúria e deve ser assinalado. Este assinalamento indica que o evento não deve ser considerado nos passos seguintes.

A próxima etapa é realizar a ordenação global dos eventos. Para esta finalidade é aplicado o algoritmo proposto em (Lamport, 1978), conhecido como relógio lógico de Lamport, descrito na Seção 2.1.2. Portanto, a ordem causal é reconstruída através do assinalamento dos eventos utilizando um contador. Ao implementar este algoritmo, os eventos locais e globais que ocorreram no sistema são ordenados seguindo a causalidade entre eles. Deste modo, a ordenação propriamente dita é feita através da atribuição de *timestamps* a cada um dos eventos. Isto permite gerar uma ordenação completa mais tênue e fácil de ser observada, para cada execução do sistema (Lanoë et al., 2019).

Tendo definido uma ordenação global para os eventos encontrados, o próximo passo da técnica de modelagem proposta é encontrar e apontar em cada evento quais são os seus eventos sucessores locais e globais, seguindo a causalidade de ocorrência. Para exemplificar, considere uma simples execução de um sistema com três processos distintos, C_1, S e C_2, apresentada na Figura 4.1. Nesta figura é possível observar 8 trocas de mensagens, gerando a ocorrência de 16 eventos identificados de a a p.

Logo, no caso da Figura 4.1, para cada evento seriam apontados os sucessores apresentados na Tabela 4.2. Neste exemplo simplificado, cada um dos eventos de envio possui dois

[1]Um impasse (*deadlock*) distribuído ocorre quando cada processo de uma coleção de processos espera que outro envie uma mensagem para ele (Coulouris et al., 2013).

sucessores: um evento global, sendo o recebimento da mensagem no nodo de destino; e um evento local, sendo o recebimento da mensagem de resposta. Por outro lado, cada um dos eventos de recebimento tem um evento de envio local como seu sucessor, com exceção do último evento da execução, o qual não possui quaisquer sucessores. Cabe lembrar que em execuções genéricas os padrões de sequências são bem mais elaborados.

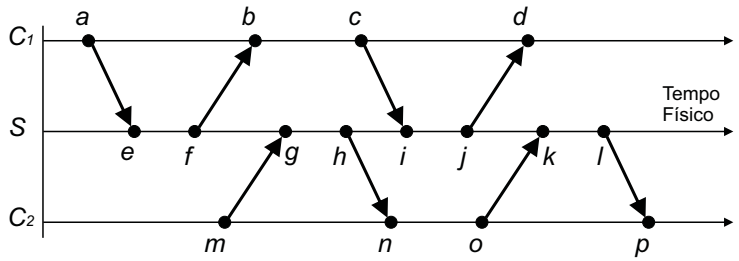


Figura 4.1: Exemplo de uma execução.

	Eventos															
Sucessor	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p
Local	b	c	d	–	f	g	h	i	j	k	l	–	n	o	p	–
Global	e	–	i	–	–	b	–	n	–	d	–	p	g	–	k	–

Tabela 4.2: Sucessores de eventos Figura 4.1

É possível interpretar estas relações de sucessão dos eventos como um grafo acíclico dirigido (*Directed Acyclic Graph* - DAG). Em um DAG, dado um vértice x qualquer, não há um caminho dirigido que comece e termine em x. Na Figura 4.2 é apresentado o DAG gerado a partir da execução demonstrada na Figura 4.1, onde cada evento é representado por um vértice e as arestas representam a relação *happened-before*.

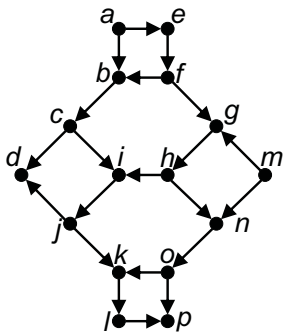


Figura 4.2: DAG gerado a partir dos eventos da Figura 4.1.

Finalmente, a partir do DAG é realizado o cálculo dos n-gramas através do algoritmo de busca em profundidade (*Depth-First Search* - DFS). Geralmente, uma execução de um sistema distribuído real contém uma grande quantidade de eventos. Portanto, neste trabalho, a busca é efetuada de modo a evitar que o mesmo caminho seja percorrido mais de uma vez, o que reduz o cálculo de n-gramas repetidos e otimiza a duração desta etapa.

Ao longo da busca realizada são calculados os n-gramas com um valor de n específico. Estendendo a Figura 4.2, na Figura 4.3 são ilustradas duas possíveis sequências de n-gramas com $n = 4$: *abcd* e *mghi*. Nesta mesma execução, existem ao todo 27 n-gramas distintos de tamanho *4*, as quais são apresentadas na Tabela 4.3. Diversas outras sequências de eventos (n-gramas) podem ser formadas através da variação do valor atribuído a n.

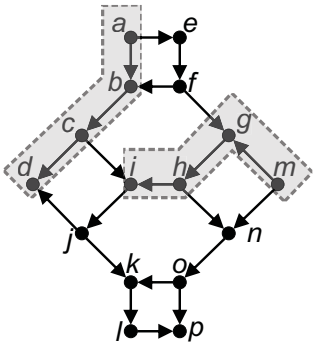


Figura 4.3: Exemplo de *n*-gramas de uma execução.

Eventos	***4*-gramas**			
a	abcd	abci	aefb	aefg
b	bcij	–	–	–
c	cijd	cijk	–	–
e	efbc	efgh	–	–
f	fbcd	fbci	fghi	fghn
g	ghij	ghno	–	–
h	hijd	hijk	hnok	hnop
i	ijkl	–	–	–
j	jklp	–	–	–
m	mghi	mghn	mnop	mnok
n	nokl	–	–	–
o	oklp	–	–	–

Tabela 4.3: Todas as possíveis sequências de *4*-gramas calculadas da execução apresentada na Figura 4.1

4.5 CONSTRUÇÃO DOS MODELOS GLOBAIS

A técnica de cálculo dos *n*-gramas apresentada na Seção 4.4 permite extrair os comportamentos possíveis de uma única execução E. Dada a natureza da base descrita na Seção 4.3 (a qual contém múltiplas execuções E), o procedimento de cálculo de *n*-gramas deve ser replicado (ou seja, repetido) para cada execução E. Somente após obter as *n*-gramas de todas as execuções é possível construir modelos globais do sistema.

Neste sentido, a técnica de modelagem proposta inclui duas estratégias, baseadas em operações elementares da teoria dos conjuntos, para construção de modelos globais. Cada estratégia tem como objetivo construir um modelo $\mathbb{M}$ que represente o comportamento normal do sistema. Logo, um modelo $\mathbb{M}$ é simplesmente um conjunto de *n*-gramas de mesmo tamanho que busca representar o comportamento correto (normal) do sistema sob estudo. Por isso, para construir um modelo só devem ser usadas as execuções corretas ($E \in \mathbb{C}$) do sistema.

A primeira estratégia proposta consiste em agrupar todos os *n*-gramas das execuções $\mathbb{E}$ que compõe a base de dados e remover as réplicas de *n*-gramas, de modo a manter apenas um exemplar de cada *n*-grama. Em suma, será construído um modelo de união $\mathbb{M} \cup (n)$ que é definido pelo conjunto dos *n*-gramas de tamanho n presentes em ao menos uma execução correta, ou seja, pela união de todos os conjuntos de *n*-gramas de tamanho n obtidos de todas as execuções corretas:

$$\mathbb{M} \cup (n) = \cup \mathbb{G}(E, n)\ \forall E \in \mathbb{C}$$

Na segunda estratégia, cada n-grama única que pertencer a todas as execuções corretas é agrupada em um único modelo, ou seja, é realizada uma interseção simples entre as execuções. Sendo assim, um modelo de interseção simples $\mathbb{M} \cap (n)$ é definido pelo conjunto dos n-gramas de tamanho n presentes em todas as execuções corretas, ou seja, pela interseção de todos os conjuntos de n-gramas de tamanho n obtidos das execuções corretas:

$$\mathbb{M} \cap (n) = \cap \mathbb{G}(E, n) \; \forall E \in \mathbb{C}$$

Uma característica almejada pela base de dados é que cada execução correta descreva um comportamento específico, ou seja, é aguardado que estas execuções sejam distintas entre si. Assim sendo, a necessidade de que um dado n-grama esteja presente em todas elas pode ser muito restritiva, resultando em um modelo pequeno (com poucos n-gramas), portanto, muito seletivo. Teoricamente, isso pode levar a um alto índice de falsos positivos nas detecções que o utilizam como base.

Uma solução possível para o excesso de seletividade do modelo de interseção simples consiste em particionar o conjunto de execuções corretas em subconjuntos. Após isso, construir modelos parciais dos mesmos usando a lógica de união. Por fim, aplicar a lógica de interseção sobre os modelos parciais construídos para obter um modelo final de interseção mais abrangente, o que chamamos de modelo de interseção amplo. De certa forma, esse o modelo de interseção amplo representa os n-gramas mais populares, pois ele contém somente os n-gramas que apareceram em todas as partições.

Explicando formalmente a construção dos modelos de interseção amplo, primeiramente, o conjunto de execuções corretas $\mathbb{C}$ é dividido em p partições $\mathbb{P}_{1...p}$ equilibradas entre si (em termos de número e tamanho das execuções de cada uma). Para cada partição $\mathbb{P}_i$ calcula-se um modelo parcial de união $\mathbb{M}'_i$ a partir das execuções E dessa partição:

$$\mathbb{M}'_i(n) = \cup \mathbb{G}(E, n) \; \forall E \in \mathbb{P}_i$$

Os modelos parciais $\mathbb{M}'_i$ então são combinados em um modelo de interseção amplo $\mathbb{M}\sqcap$, usando uma lógica de interseção; dessa forma, esse modelo conterá os n-gramas presentes em todos os modelos parciais:

$$\mathbb{M} \sqcap (n) = \cap \mathbb{M}'_i(n) \; \forall i \in \{1...p\}$$

De certa forma, o modelo amplo $\mathbb{M}\sqcap$ engloba os demais, sendo um híbrido entre os modelos de união $\mathbb{M}\cup$ e de interseção simples $\mathbb{M}\cap$ e pode representar ambos: se for feita uma só partição ($p = 1$) temos $\mathbb{M}\sqcap = \mathbb{M}\cup$; se for feita uma partição para cada execução correta ($p = |\mathbb{C}|$) temos $\mathbb{M}\sqcap = \mathbb{M}\cap$. A escolha do valor de p ($1 \leq p \leq |E|$) é um parâmetro a ser avaliado em trabalhos futuros.

Consideramos neste trabalho, que cada n-grama do modelo representa uma pequena parte do comportamento correto do sistema. Desta forma, a fase de construção dos modelos é realizada de forma *offline*, bem como a fase de detecção. Logo, a detecção *offline* de anomalias usando um modelo $\mathbb{M}$ é realizada da seguinte forma:

1. Dada uma execução E a ser analisada, é extraído o conjunto de n-gramas $\mathbb{G}(E, n)$ da mesma.

2. Se todos os n-gramas de $\mathbb{G}(E, n)$ estiverem presentes no modelo $\mathbb{M}$ (ou seja, se $\mathbb{G}$ estiver contido em $\mathbb{M}$), a execução é considerada normal; caso contrário, é anômala:

$$\mathbb{G}(E, n) \subseteq \mathbb{M}? \begin{cases} \textit{sim} & \rightarrow \text{execução correta} \\ \textit{não} & \rightarrow \text{execução anômala} \end{cases}$$

Em trabalhos futuros, pretende-se investigar a detecção *online* de anomalias. Para isso, tendo como base o uso de um modelo $\mathbb{M}$ pretende-se implementar a detecção no seguinte formato: dada uma execução E em andamento a ser analisada, um *processo observador*, capaz de observar todos os eventos do sistema, manterá uma lista L com os últimos n eventos observados, em ordem de ocorrência: $L = [e_1, e_2, \ldots, e_n]$. A lista L pode ser vista como um n-grama e testada usando o modelo $\mathbb{M}$:

$$L \in \mathbb{M}? \begin{cases} sim & \rightarrow \text{comportamento correto} \\ não & \rightarrow \text{comportamento anômalo} \end{cases}$$

A cada novo evento observado no sistema, a lista L deve ser atualizada, descartando o evento mais antigo e adicionando o novo evento, e testada novamente. Ressaltando que essa definição foi apresentada de uma forma bem generalizada e, existem inúmeros desafios em torno de se realizar a detecção *online* de anomalias.

4.6 CONSIDERAÇÕES FINAIS

Este trabalho apresenta uma técnica para a construção de modelos de comportamento normal de sistemas distribuídos, baseando-se na utilização de n-gramas. Cada n-grama representa uma parte de um estado do sistema, ou seja, atividades corretas que podem acontecer no sistema modelado. Para isso, os n-gramas são calculados a partir de uma base de eventos corretos extraídos do sistema. A ordenação dos eventos é realizada através da utilização de relógios lógicos, ou seja, eventos locais e globais são ordenados seguindo a causalidade entre eles. O próximo passo é a construção de um grafo acíclico dirigido (DAG), que aponta os sucessores de cada evento. O cálculo das gramas é realizado através de uma busca em profundidade sob o DAG, respeitando a ordem de causalidade.

Tendo calculado as gramas, os modelos globais são construídos baseados em operações elementares da teoria dos conjuntos. A primeira estratégia adotada é unir todos os n-gramas presentes nas execuções que compõe a base de dados, removendo as réplicas. Para a segunda estratégia, dois formatos de interseção foram propostos devido às características da base de dados: a construção de modelos de interseção simples, onde cada n-grama que pertencer às execuções corretas é agrupada em um único modelo; e, a construção de modelos de interseção amplo, onde $\mathbb{E}$ é dividido em partições, através das partições são construídos modelos parciais e então, usando a lógica de interseção, esses são combinados de forma a extrair somente os n-gramas que estão contidos em todos os modelos parciais, gerando o modelo final.

Os modelos construídos são empregados durante o processo de detecção de anomalias. Para isso são calculadas as gramas de um traço a ser analisado, para que T seja classificado como correto suas gramas devem ser conhecidas, ou seja, $\mathbb{G}(T, n) \subseteq \mathbb{M}(n)$. Caso contrário T é classificado como suspeito.

5 VALIDAÇÃO DA PROPOSTA

Neste capítulo será apresentada uma avaliação da técnica proposta. Para este fim, foi implementado um protótipo seguindo todos os passos descritos no Capítulo 4. Inicialmente na Seção 5.1, a base de dados utilizada para o estudo é descrita, contendo detalhes acerca da aplicação utilizada para sua produção, bem como os cenários das execuções corretas e anômalas que foram registradas. Os procedimentos adotados para a construção dos modelos, bem como as métricas aplicadas nas avaliações, são detalhadas na Seção 5.2. Por fim, a Seção 5.3 discorre sobre os resultados alcançados.

5.1 CENÁRIO E MÉTODO DE COLETA DOS TRAÇOS

A avaliação da proposta consiste na construção de um conjunto de modelos a partir de dados de uma aplicação distribuída real. A base de dados utilizada no estudo é constituída por traços de execuções de uma aplicação distribuída, produzida no contexto descrito em (Lanoë et al., 2019), e concedida pelo mesmo. Essa base ainda não é pública, mas foi escolhida por apresentar múltiplas execuções de uma aplicação real, que são representados por traços de eventos que ocorreram em cada nodo da aplicação.

O cenário de avaliação usa como aplicação distribuída o XtreemFS[1], que provê um sistema de arquivos distribuído, replicado e tolerante a falhas (Quobyte Inc, 2020). Uma implantação do XtreemFS é composta por vários servidores que podem ser executados em diferentes nodos, interconectados em uma rede local ou mesmo pela Internet. Assim, desde que os servidores estejam acessíveis, um módulo cliente instalado em qualquer máquina do mundo pode acessar o sistema.

Além da característica distribuída, esta plataforma oferece outros recursos interessantes como: (I) *data striping* (conhecido também como RAID 0, que permite a leitura e gravação de dados distribuídos simultaneamente entre discos do arranjo); (II) replicação (todos os seus componentes podem ser replicados para redundância, que resulta em um sistema de arquivos tolerante a falhas), incluindo o suporte a réplicas parciais que são preenchidas apenas com o conteúdo sob demanda; (III) mecanismos para autenticação e autorização do usuário, além da possibilidade de criptografar o tráfego de rede (Kolbeck et al., 2015).

Para construir o ambiente necessário para prover o sistema de arquivos distribuído, o XtreemFS é formado por quatro componentes:

- OSD (*Object Storage Device*): entidade encarregada de armazenar o conteúdo real dos arquivos, podendo ser replicada em diferentes nodos.

- MRC (*Metadata and Replica Catalog*): armazena as informações sobre os volumes de dados e gerencia a árvore de diretórios e os metadados de cada arquivo (como nome, tamanho e data de modificação). Este elemento também é responsável pelo controle de acesso aos arquivos e a autenticação dos usuários.

- DIR (*Directory Service*): componente que registra todos os serviços do sistema, incluindo um banco de dados de serviços e volumes disponíveis. Este elemento conecta todos os demais componentes do sistema. O MRC o utiliza para descobrir servidores de armazenamento.

[1]Mais informações sobre o XTreemFS podem ser obtidas no site: http://www.xtreemfs.org/

- Clientes: aplicação cliente que permite aos usuários realizarem requisições ao sistema e gerenciarem os estados dos arquivos, executando operações como criação/montagem de volumes, criação de arquivos, etc.

A implantação do XtreemFS realizada por (Lanoë et al., 2019) foi preparada contendo todos estes componentes. A comunicação entre os componentes foi realizada através da troca de mensagens via rede, sobre um canal de comunicação confiável, viabilizado através do protocolo TCP.

A partir do cenário construído, foram efetuadas diversas execuções do XtreemFS, no intuito de coletar traços de execuções que representam diferentes comportamentos normais do sistema, bem como o seu comportamento sob ataques. Ao decorrer de todo o procedimento de coleta de registros dos eventos, foi alterado o tempo de duração da execução, entre 1 minuto até 5 horas. Da mesma forma, a quantidade de nodos do sistema também varia, onde em algumas delas podem existir até dois clientes ativos (realizando solicitações a aplicação) e em outras nenhum. Por fim, os traços coletados por cada processo tem um tamanho médio de 39 KB, estruturados em cerca de 130 execuções.

Para a obtenção da base, todos os nodos do sistema foram configurados para registrar seus traços individuais, contendo informações sobre eventos de envio ou recepção de mensagens. Nenhuma informação em relação ao horário físico local foi armazenada juntamente aos eventos, para que seja posteriormente observada a causalidade. Os eventos foram armazenados em uma ordem FIFO, ou seja, a ordem de registro no traço segue a ordem de ocorrência local. Logo, a relação entre um evento de envio e um evento de recebimento pode ser realizada facilmente e, a ordenação parcial entre os eventos de cada execução pode ser induzida conforme o decorrer da leitura dos traços de cada nodo (Totel et al., 2016).

Para cada execução E um novo traço T por processo foi registrado (T_1, T_2, .., T_j), conforme o conjunto de vezes (1, 2, . . ., i) que a aplicação foi executada. Esta representação de traços T_{ij} é similar a um conjunto de *logs*. É importante destacar que durante a fase de coleta de traços de comportamento normal é assumido que nenhum comportamento anormal ou ataque tenha ocorrido (Lanoë et al., 2019).

Cada evento e_{ij} registrado contém os seguintes campos: *Source IP* armazena o endereço IP do nodo que deu origem ao evento; *Destination IP* indica o endereço do nodo de destino; *Type* informa se a mensagem corresponde a um envio (!) ou recepção (?); o último campo, *Data*, sinaliza a natureza do evento, podendo ser usado para identificar se determinado evento é uma requisição ou uma resposta. A Figura 5.1 apresenta a estrutura desses registros para uma execução hipotética entre dois clientes e um servidor, os eventos são os mesmos apresentados na Figura 4.1.

Nó "servidor S" (`IP1`)

```
IP2 - IP1 ? consulta
IP3 - IP1 ? consulta
IP1 - IP2 ! disponível
IP1 - IP3 ! disponível
IP2 - IP1 ? adquirir
IP3 - IP1 ? adquirir
IP1 - IP2 ! venda
IP1 - IP3 ! venda
```

Nó "cliente C_1" (`IP2`)

```
IP2 - IP1 ! consulta
IP1 - IP2 ? disponível
IP2 - IP1 ! adquirir
IP1 - IP2 ? venda
```

Nó "cliente C_2" (`IP3`)

```
IP3 - IP1 ! consulta
IP1 - IP3 ? disponível
IP3 - IP1 ! adquirir
IP1 - IP3 ? venda
```

Figura 5.1: Exemplo de traços de uma aplicação distribuída.

Objetivando coletar traços de execuções com comportamentos anômalos do sistema, uma versão não segura do XtreemFS foi implementada por (Lanoë et al., 2019). Este conjunto de execuções anômalas é definido por $\mathbb{A}$, logo $\mathbb{A} \subseteq \mathbb{E}$ define o subconjunto de execuções anômalas (ataques) presentes em $\mathbb{E}$. Quatro ataques conhecidos contra a sua integridade foram realizados: o ataque *NewFile*, que adiciona metadados de um arquivo no servidor MRC e não insere seu conteúdo no servidor OSD; *DeleteFile*, que exclui os metadados do arquivo no servidor MRC e mantém seu conteúdo no servidor OSD; *Chmod*, que altera a política de acesso a arquivos no servidor MRC, mesmo sem a autorização; e, por último, o ataque *Chown*, que modifica o proprietário do arquivo nos metadados MRC.

Cada um dos ataques foi executado em quatro contextos diferentes: (c1) nenhum cliente ativo; (c2) com clientes ativos no ambiente, mas antes de realizarem operações; (c3) após as operações serem realizadas e; (c4) originado de um endereço que não pertence ao cliente. Cerca de 16% dos traços da base representam estes comportamentos anômalos. Os traços dessas execuções permitem a avaliação da eficácia de detecção de cada modelo previamente construído a partir do comportamento normal do sistema. Contudo, um trabalho futuro previsto é aumentar o número de execuções normais e de ataques na base de traços, buscando obter uma representação mais ampla e equilibrada dos comportamentos possíveis da aplicação em questão.

5.2 MODELOS DE VALIDAÇÃO

Com o objetivo de avaliar a técnica apresentada no Capítulo 4, bem como os modelos gerados, foi implementado um protótipo que contempla todos os passos descritos. Como resultado, foi construído um conjunto de modelos de normalidade. Nesta seção será descrito como, de fato, ocorreu a construção dos modelos, bem como as métricas utilizadas para sua avaliação. A Tabela 5.1 apresenta as notações adicionais que serão utilizadas para esta validação.

Notação	**Descrição**
$\mathbb{A}$	subconjunto de execuções anômalas (ataques) presentes em $\mathbb{E}$..
$\mathbb{P}V_p$	partições de $\mathbb{C}$ para validação.
$\mathbb{M}V$	modelos diversificados de $\mathbb{P}V_p$ para validação.
$\mathbb{M}V_i$	um modelo para validação.
$\mathbb{M}V\cup$	todos os modelos de união, pertencentes a $\mathbb{M}V$ para validação.
$\mathbb{M}V\cap$	todos os modelos de interseção simples, pertencentes a $\mathbb{M}V$ para validação.
$\mathbb{M}V\sqcap$	todos os modelos de interseção amplo, pertencentes a $\mathbb{M}V$ para validação.
$\mathbb{M}V\cup_i$	um modelo de união para validação.
$\mathbb{M}V\cap_i$	um modelo de interseção simples para validação.
$\mathbb{M}V\sqcap_i$	um modelo de interseção amplo para validação.

Tabela 5.1: Notações adicionais empregadas durante a validação dos modelos.

Para cada execução E correta presente na base, foram calculadas gramas com diferentes valores de n, de 3 a 15. Gramas menores que 3 não são capazes de descrever um comportamento correto adequadamente, e maiores que o valor selecionado demonstraram um custo computacional elevado durante as etapas de construção dos modelos e detecção de anomalias.

Para que seja possível avaliar a eficácia dos modelos construídos (união e interseção), é necessário analisar a sua capacidade de classificação diante de execuções corretas que não foram utilizadas durante a sua construção. Para isso, após obter as gramas de todas as execuções, o conjunto de traços corretos $\mathbb{C}$ foi separado em dados de treinamento e validação. Logo, $\mathbb{C}$ foi segmentado em p partições de validação $\mathbb{P}V_p$, $p = 1 \ldots p$ individuais. As execuções foram

distribuídas em cada $\mathbb{P}V_p$ de forma a equilibrar o tamanho e a quantidade de traços. Cada $\mathbb{P}V_p$ será utilizada tanto na construção dos modelos quanto para avaliar o processo de detecção de anomalias. Em suma, $\mathbb{P}V_p$, $\mathbb{C}$ e $\mathbb{A}$ são *partições*[2] de $\mathbb{E}$, ou seja, $|\mathbb{P}V_p| = \mathbb{C}$, $\mathbb{C} \cup \mathbb{A} = \mathbb{E}$ e $\mathbb{C} \cap \mathbb{A} = \emptyset$.

Buscando aumentar a eficiência do algoritmo de criação dos modelos, foi criado um arquivo único contendo todas as gramas encontradas nos traços que compõem cada partição, ou seja, foram construídos modelos parciais de união para validação $\mathbb{M}V'_i$ a partir das execuções E de cada partição.

Em seguida, os modelos de validação $\mathbb{M}V$ foram gerados em um esquema de validação cruzada k-fold (Arlot et al., 2010). Este k possui o mesmo valor atribuído a p. Para as avaliações presentes neste trabalho consideramos $k = 5$ (ou $p = 5$). Nesse esquema, uma partição de execuções é separada para validação, enquanto $k - 1$ partições são combinadas para produzir os modelos, resultando em k modelos de validação ($\mathbb{M}V_{1...k}$). Consequentemente, as partições foram combinadas seguindo a lógica de união e interseção, conforme descrito na Seção 4.5.

	Quantidade média de *n*-gramas				
N	E	$\mathbb{P}V$	$\mathbb{M}V\cup$	$\mathbb{M}V\cap$	$\mathbb{M}V\sqcap$.
3	970	1.032	1.398	94	703.
6	2.790	6.819	10.387	142	3.582.
9	7.369	28.716	47.497	122	11.929.
12	20.980	105.712	185.372	96	36.243.
15	55.752	353.744	650.324	38	101.242 .

Tabela 5.2: Número médio de n-gramas calculados em função de n.

A Tabela 5.2 informa o número médio de n-gramas obtidos durante as etapas de construção dos modelos. Observa-se claramente que o número de n-gramas cresce com o valor de n, em uma razão sub-exponencial. Portanto, modelos gerados com n-gramas mais longos são maiores (têm mais n-gramas), podendo representar mais comportamentos. Outro ponto relevante é que a estratégia adotada para construir $\mathbb{M}V\cup$ resultou em modelos extremamente pequenos, tornando-os insignificantes para a detecção de anomalias.

Durante a fase de detecção de anomalias, estes modelos são utilizados como base para distinguir um comportamento anômalo de um correto. Dessa forma, se um traço T contiver apenas n-gramas presentes em $\mathbb{M}V(n)$, T é classificado como correto; porém, se T contiver n-gramas desconhecidas por $\mathbb{M}V(n)$, T é classificado como suspeito. Ou seja, se $\mathbb{G}(T, n) \subseteq \mathbb{M}V(n)$ então T é correto diante de $\mathbb{M}V(n)$; senão, T é suspeito diante de $\mathbb{M}V(n)$.

Para avaliar $\mathbb{M}V\cup$ e $\mathbb{M}V\sqcap$, cada $\mathbb{M}V$ é testado usando todos os traços corretos $\mathbb{C}$ que não foram adotados para o compor, separados durante a construção dos modelos. O resultado retornado pelo teste de cada $\mathbb{M}V$ é o percentual de n-gramas encontrados nos modelos em relação ao total de n-gramas presentes no traço analisado.

Baseado nisso, o traço será classificado como correto ou suspeito. Como se trata de uma classificação binária, existem quatro condições para essa classificação:

(I) Verdadeiro Positivo (VP): Quando T representa uma anomalia e $\mathbb{M}V$ o classifica como uma anomalia;

(II) Verdadeiro Negativo (VN): Quando T equivale a um comportamento correto e $\mathbb{M}V$ o classifica corretamente;

[2]Um conjunto X pode ser dividido em subconjuntos disjuntos denominados *partições* P_i de forma que $\forall i \cup P_i = X$ e $\forall(i, j)\ P_i \cap P_j = \emptyset$.

(III) Falso Positivo (FP): Quando T se trata de um comportamento correto e $\mathbb{M}V$ o classifica erroneamente como uma anomalia;

(IV) Falso Negativo (FN): Quando T corresponde a uma anomalia e $\mathbb{M}V$ o classifica erroneamente como um comportamento correto.

O modelo perfeito seria obtido quando todas as classificações fossem realizadas corretamente, ou seja, nenhum FP e FN seria retornado. Porém, esse grau de eficácia é praticamente inatingível. Sendo assim, podemos utilizar os números de cada condição para definir métricas de eficácia para os modelos construídos.

Buscando analisar o custo de cada modelo, a primeira avaliação é dada através das taxas de FP ($\mathbb{T}_{FP}$). Seu cálculo é realizado empregando a divisão entre a quantidade de traços corretos classificados incorretamente como uma anomalia pela quantidade total de traços corretos $\mathbb{C}$.

$$\mathbb{T}_{FP} = \frac{FP}{\mathbb{C}} \tag{5.1}$$

Para analisar a fração de amostras positivas classificadas corretamente, é importante verificar as taxas de VP, conhecido também como sensibilidade ou *recall*. Por meio dessa análise é possível avaliar o quão íntegra está a classificação realizada pelo modelo. Sua fórmula é dada pela razão entre a quantidade de T anômalos corretamente identificados (VP) e o total de traços anômalos usados.

$$\mathbb{T}_{VP} = \frac{VP}{VP + FN} \tag{5.2}$$

A precisão dos modelos é avaliada a partir da divisão entre a quantidade de traços positivos identificados corretamente pelo total de traços classificados como anômalos. Esta análise permite verificar quão benéfico estão os modelos construídos.

$$\mathbb{P} = \frac{VP}{VP + FP} \tag{5.3}$$

Por fim, umas das métricas mais relevantes quando se trata de análise de modelos e de IDSs é a acurácia. De forma geral, acurácia representa o quão próximo o resultado retornado é do seu valor de referência real. Sendo assim, para considerarmos que o resultado é satisfatório ele deve estar acima de 50%, no mínimo. O seu cálculo é realizado a partir da razão entre as classificações corretas pelo total de amostras utilizadas.

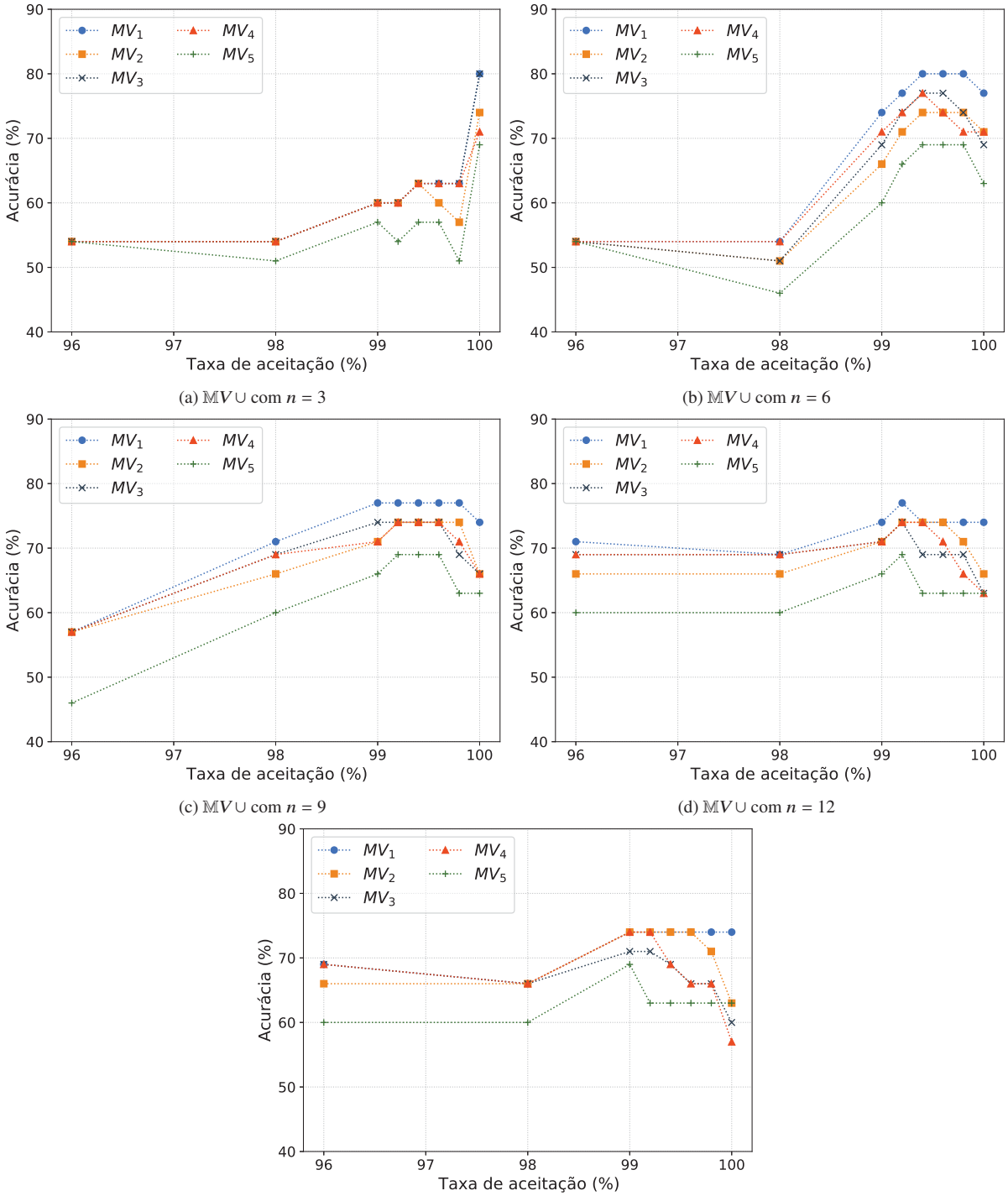
$$Ac = \frac{VP + VN}{\mathbb{E}} \tag{5.4}$$

É necessário pontuar a diferença entre precisão e acurácia. A precisão é relacionada à variação entre as classificações obtidas, ou seja, o quão dispersos esses resultados podem estar entre si. Já a acurácia indica a distância dos resultados obtidos baseado no seu valor real.

5.3 RESULTADOS E ANÁLISE

Pelo fato dos modelos terem sido construídos através de uma base de dados, há uma probabilidade significativa de que não sejam totalmente representativos, em outras palavras, não possuam todos os comportamentos normais da aplicação. Como a relação $\subseteq$ só define se caso um traço T pertence ou não ao modelo, ou seja, de forma binária (correto ou suspeito), adicionando o fato anterior, se torna necessário adotar um critério menos rígido. Por isso,

foi adotada uma *taxa de aceitação*, buscando flexibilizar a classificação de comportamentos normais e anômalos. Assim, foi considerado como resultado retornado pelo teste de cada $\mathbb{M}V$, o percentual de n-gramas presentes no traço analisado que também se encontram nos modelos.



(a) $\mathbb{M}V\cup$ com $n = 3$

(b) $\mathbb{M}V\cup$ com $n = 6$

(c) $\mathbb{M}V\cup$ com $n = 9$

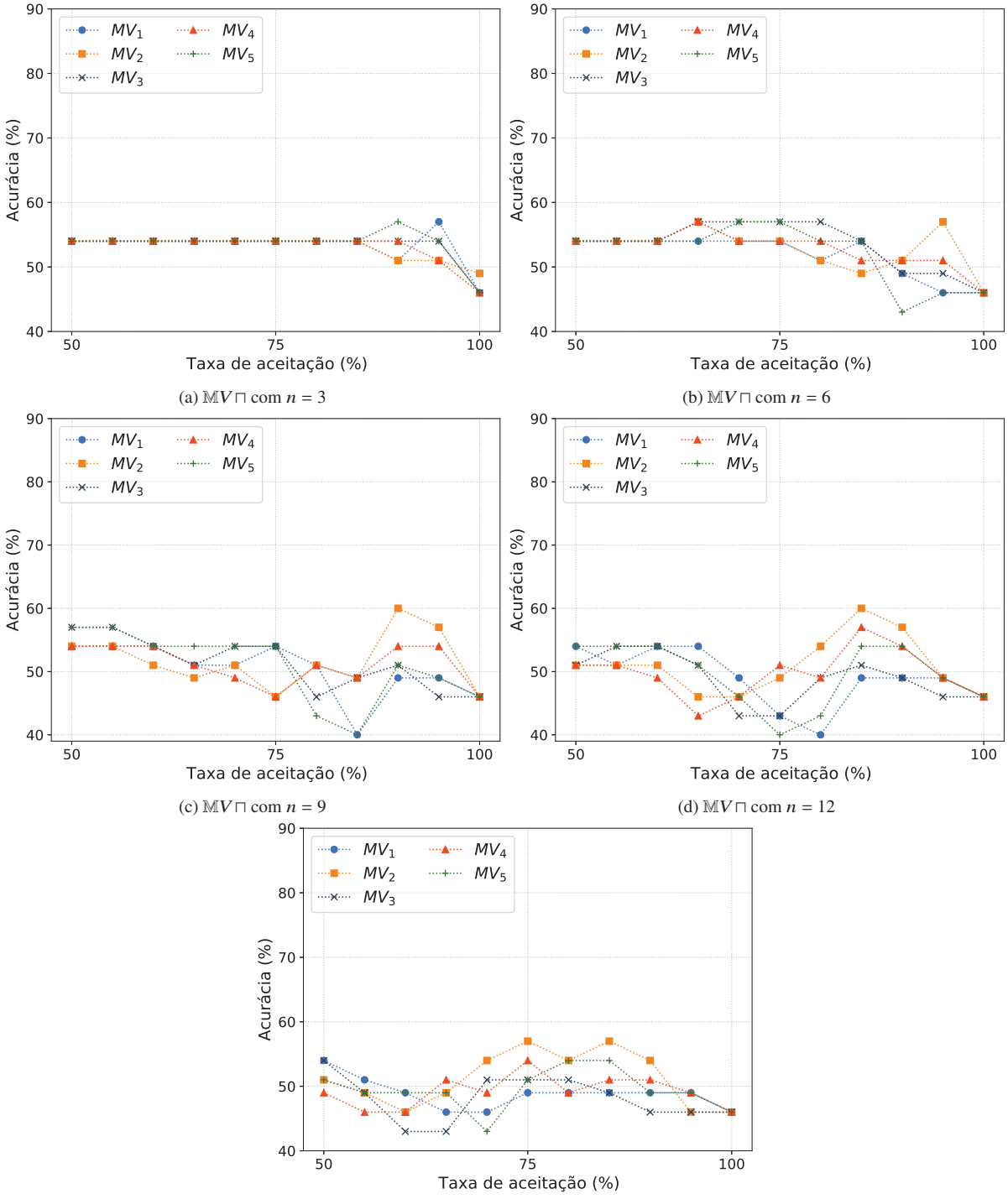
(d) $\mathbb{M}V\cup$ com $n = 12$

(e) $\mathbb{M}V\cup$ com $n = 15$

Figura 5.2: Acurácia dos modelos de união $\mathbb{M}V\cup$ variando o tamanho de n

Para compreender qual a taxa de aceitação mais adequada para cada tipo de modelo, foi verificado individualmente o comportamento da acurácia conforme a variação da taxa de aceitação. A figura 5.2 representa os modelos de validação de união $\mathbb{M}V\cup$. As taxas de aceitação apresentadas variam de 96% a 100%, valores menores resultaram em uma acurácia $\leq$ 54%. De

forma geral, a melhor acurácia obtida para esses modelos foi com $n = 3$ e 6 (5.2(a) e 5.2(b)), chegando a 80%, porém, com diferentes taxas de aceitação. A maior oscilação notada foi entre 99% e 100%, onde em 3 de 5 casos (5.2(c), 5.2(d), 5.2(e)) a acurácia diminuiu em média 8 pontos percentuais. Isso ocorre porque ao exigir a relação de pertinência total entre o T e $\mathbb{M}V\cup$, exigimos que $\mathbb{M}V\cup$ contenha todas $\mathbb{G}(T, n)$, consequentemente diminuímos sua capacidade de detecção. Sendo assim, uma análise minuciosa entre esses valores foi realizada, possibilitando observar que a taxa de aceitação adequada para esses modelos é de 99,4%. Dessa forma, se um traço T tiver 99,4% de suas gramas conhecidas, ele é considerado normal perante $\mathbb{M}V\cup$.



(a) $\mathbb{M}V\sqcap$ com $n = 3$

(b) $\mathbb{M}V\sqcap$ com $n = 6$

(c) $\mathbb{M}V\sqcap$ com $n = 9$

(d) $\mathbb{M}V\sqcap$ com $n = 12$

(e) $\mathbb{M}V\sqcap$ com $n = 15$

Figura 5.3: Acurácia dos modelos de interseção amplo $\mathbb{M}V\sqcap$ variando o tamanho de n

Observando a individualidade de cada modelo $\mathbb{M}V\cup_k$ conforme a taxa de aceitação selecionada, o que apresentou a menor acurácia foi $\mathbb{M}V\cup_5$, atingindo uma média geral de 61%. Com uma diferença de 10 pontos percentuais, $\mathbb{M}V\cup_1$ demonstrou ser o modelo mais representativo, sendo o que alcançou a melhor acurácia para todas as variações de n.

A mesma avaliação foi realizada para os modelos de validação de interseção amplo $\mathbb{M}V\sqcap$, demonstrada na figura 5.3. Neste cenário, a acurácia foi menor se comparada a $\mathbb{M}V\cup$, sendo 60% o valor máximo atingido, demonstrado por $\mathbb{M}V\sqcap_2$ para $n = 9$ e 12 (5.3(c) e 5.3(d)). Para uma taxa de aceitação de 100% ($T \subseteq MV\sqcap$), em praticamente todos os casos a acurácia atingida foi de 46%. Isto ocorreu devido a esses modelos classificarem quaisquer T como suspeito, ou seja, traços anômalos foram classificados corretamente, porém, todos os traços corretos foram classificados erroneamente como suspeitos. De outro modo, para uma taxa de aceitação $\leq$ 55% os modelos classificam quaisquer T como normal. Sendo assim, a taxa de aceitação adotada para estes modelos foi de 75%.

Encontrada uma taxa de aceitação adequada para ambos os modelos construídos, a próxima avaliação refere-se às taxas de falsos positivos ($\mathbb{T}_{FP}$). Para esta avaliação e posteriores, os resultados apresentados serão uma média aritmética dos k modelos de validação construídos, prática comumente adotada para a validação cruzada k-fold. A figura 5.4 exibe a $\mathbb{T}_{FP}$ alcançada pelos $\mathbb{M}V$ conforme os valores de n. Seguindo o esperado, $\mathbb{M}V\sqcap$ apresentou a maior taxa se comparado a $\mathbb{M}V\cup$, atingindo 54% de diferença em $n = 15$. Isso evidência que coletar somente as gramas mais populares gera modelos muito restritos e incapazes de distinguir um comportamento correto de uma anomalia. Outro ponto que pode ser observado na imagem é que quanto maior o valor de n, mais específicos se tornam os n-gramas, consequentemente, a quantidade de comportamentos representados é reduzida, tornando os modelos mais propensos a erros.

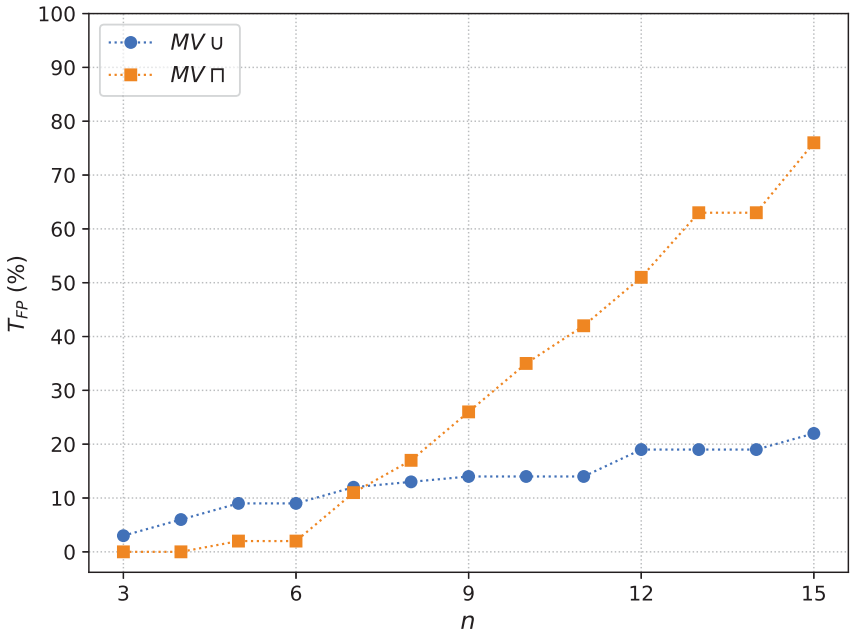


Figura 5.4: Taxa de falsos positivos para diferentes valores de n em $\mathbb{M}V\cup$ e $\mathbb{M}V\sqcap$

Buscando analisar o quão satisfatório estão os resultados retornados pelos modelos construídos, a taxa de verdadeiro positivo $\mathbb{T}_{VP}$ foi verificada e os resultados retornados são exibidos na figura 5.5. Um primeiro ponto a ser observado é que valores de n pequenos (≤ 4) não são capazes de representar adequadamente um comportamento (apenas $\approx$ 20% dos traços foram classificados corretamente), classificando a maioria dos T incorretamente como uma atividade normal. Ainda analisando a variação de tamanho dos n-gramas, $\mathbb{M}V\sqcap$ evolui sua capacidade de

detecção significativamente conforme aumentam os valores atribuídos a n. Este comportamento indica que $\mathbb{M}V\sqcap$ permite a identificação de ataques com mais facilidade, mas, em contrapartida, cresce a dificuldade de identificar corretamente os traços normais. Em trabalhos futuros, uma investigação buscando flexibilizar a combinação dos modelos de interseção e construir modelos menos restritos será realizada.

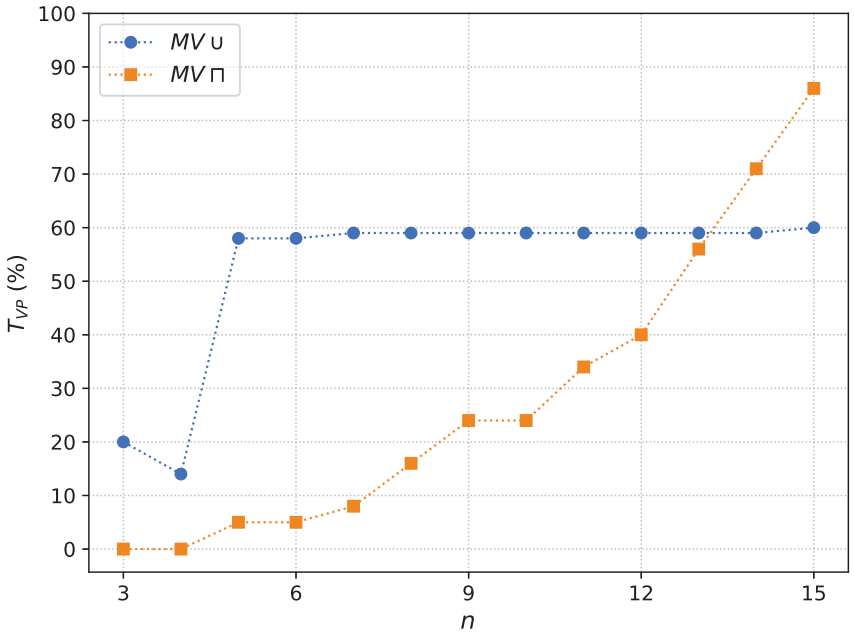


Figura 5.5: Taxa de verdadeiro positivo para diferentes valores de n em $\mathbb{M}V\cup$ e $\mathbb{M}V\sqcap$

Neste cenário, $\mathbb{M}V\cup$ demonstrou ser estável independentemente da variação de n, permanecendo com $\mathbb{T}_{VP} \approx 58\%$. Esta estabilidade representa que as n-gramas geradas com $n > 6$ são pouco relevantes para a detecção de ataques. A tabela 5.3 apresenta a capacidade de detecção para estes modelos. Como apresentado na Seção 5.1, os quatro tipos de ataques representados na base de dados são considerados. Na tabela, uma indicação $x/5$ indica que x dos 5 modelos detectaram o ataque; por clareza, $0/5$ é indicado como "-" e $5/5$ por "✓". Observando especificamente o ataque *Chown*, somente um dos cinco modelos construídos foi capaz de detectá-lo em c4 (originado de um endereço que não pertence ao cliente), independente do tamanho das gramas. Para o mesmo ataque em outros contextos, dois outros modelos distintos o identificaram como uma possível ameaça.

	Ataque	*NewFile*				*DeleteFile*				*Chmod*				*Chown*			
N	**Contexto**	c1	c2	c3	c4	c1	c2	c3	c4	c1	c2	c3	c4	c1	c2	c3	c4.
6	$\mathbb{M}V\cup$	-	✓	✓	✓	-	✓	✓	✓	-	✓	✓	✓	-	-	-	1/5
12		-	✓	✓	✓	-	✓	✓	✓	-	✓	✓	✓	-	-	1/5	1/5
15		-	✓	✓	✓	-	✓	✓	✓	-	✓	✓	✓	-	1/5	1/5	1/5 .

Tabela 5.3: Detecção de verdadeiro-positivo por cada modelo.

A tabela 5.3 mostra a dificuldade de detecção de c1 (nenhum cliente ativo), onde em nenhum dos ataques os traços foram dados como suspeitos. Isso evidencia dois pontos: em primeiro lugar é provável que os traços usados para a construção dos modelos não contenham um número suficiente de comportamentos normais dentro desse mesmo contexto (porém, sem o ataque) para representar adequadamente o sistema. O segundo ponto é que os modelos são

capazes de detectar somente anomalias em cenários em que existam clientes ativos, traços anormais que antecedem atividades corretas de clientes são classificados como comportamentos normais. Por outro lado, a capacidade de detecção foi satisfatória, já que com um tamanho de n relativamente pequeno os resultados são similares a gramas maiores e todo o custo de processamento pode ser evitado.

Por fim, se faz necessário averiguar o quão preciso estão os modelos, ou seja, qual a relevância das classificações que os utilizam como base. A figura 5.6 demonstra a precisão ($\mathbb{P}$) obtida por cada modelo em função de n. Analisando $\mathbb{M}V\cup$ é possível afirmar que o aumento no tamanho dos n-gramas não é uma boa estratégia, pois a quantidade de resultados incorretos tendem a crescer. Já para $\mathbb{M}V\sqcap$ não é possível realizar a mesma afirmação, visto que seu comportamento se manteve estável na maioria dos casos testados.

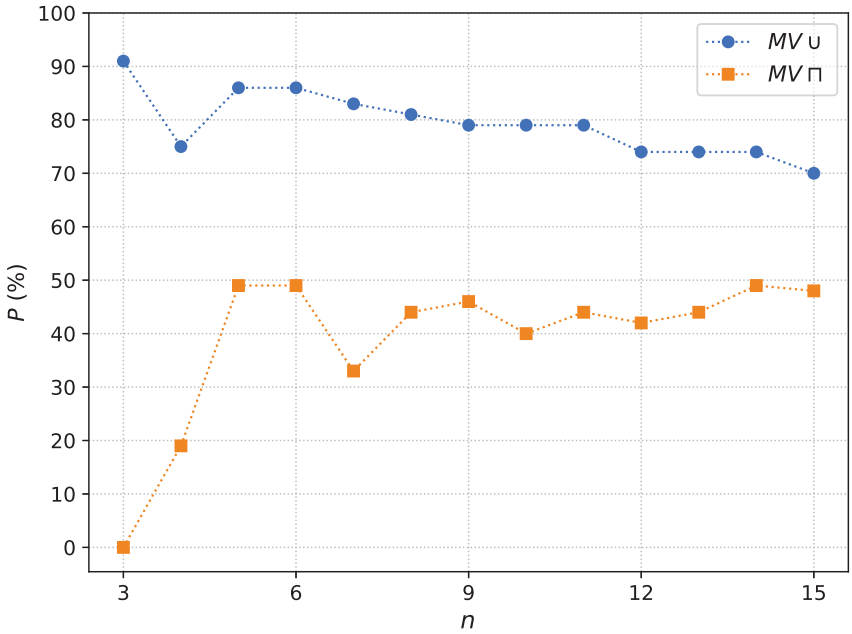


Figura 5.6: Precisão para diferentes valores de n em $\mathbb{M}V\cup$ e $\mathbb{M}V\sqcap$

Para este cenário, o tamanho mais adequado para os n-gramas em modelos baseados em união é de $n = 6$, considerando todas as análises realizadas. Sendo que: apresentou a melhor acurácia (75%, apresentada na Figura 5.2) comparada aos demais tamanhos; a $\mathbb{T}_{FP}$ é razoável (9%, apresentada na Figura 5.4), além de estar entre as menores retornadas pelo modelo; sua capacidade de detecção ($\mathbb{T}_{VP}$) é similar a de n-gramas maiores, como demonstrado na Figura 5.5; e, a precisão dos resultados é satisfatória (86%, exibida na Figura 5.6), o que significa que são retornados resultados mais relevantes que irrelevantes. Lembrando que este valor é cabível para este cenário de avaliação, caso a técnica seja aplicada em bases ou cenários diferentes, este valor pode variar.

5.4 CONSIDERAÇÕES FINAIS

Para a avaliação da técnica proposta, foi utilizada uma base de dados contendo eventos de execuções de uma aplicação distribuída real, que foi construída e fornecida por (Lanoë et al., 2019). Esta base contêm execuções do XtreemFS, que fornece um sistema de arquivos distribuído, composto por vários servidores que foram executados em diferentes nodos. Cada execução é formada por traços individuais dos nodos. Os traços armazenam eventos de envio e recepção

de mensagens que ocorreram durante a execução. Essas execuções representam situações de comportamento normal do sistema e também do sistema sob ataque.

Um protótipo da técnica foi implementado, seguindo todos os passos descritos. Todas as execuções presentes na base tiveram suas gramas calculadas, variando o tamanho de n, de 3 a 15. Logo, $\mathbb{E}$ foi dividida em k partições de validação para que os modelos pudessem ser construídos em um esquema de validação cruzada k-fold. Sendo assim, foram gerados k modelos de validação $\mathbb{M}V_i$, cada um usando $k-1$ partições. Em seguida, os subconjuntos foram combinados de duas formas, por união e por interseção. Devido à interseção das execuções gerar modelos pequenos e incapazes de representar o sistema corretamente, um terceiro conjunto de modelos foi gerado, baseado na interseção das partições de validação.

Com base nas métricas selecionadas para avaliar os modelos, foi possível verificar que a estratégia de união gera modelos com boa capacidade de diferenciar um comportamento correto de uma anomalia. Por outro lado, ambas as estratégias de interseção geram modelos restritos, tendendo a classificar a maior parte dos comportamentos como anômalos. Em relação aos tamanhos dos n-gramas, nota-se que tamanhos maiores de n são mais efetivos para a detecção de ataques, devido a representarem comportamentos mais específicos.

6 CONCLUSÃO

Este trabalho propôs uma técnica para a construção de modelos de normalidade utilizando como base *n*-gramas de eventos. Foram descritos os processos para a obtenção dos *n*-gramas através dos traços de execução de uma aplicação distribuída real e a construção de modelos de normalidade do sistema, através de operações sobre os conjuntos. A utilização de *n*-gramas para a construção de modelos de normalidade é uma prática comum em ambientes web, porém, até onde sabemos, esta é a primeira proposta que utiliza *n*-gramas para modelar um sistema distribuído. Resultados iniciais desta pesquisa foram publicados na trilha principal do XX Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais (SBSeg) (Viescinski et al., 2020).

A técnica proposta não depende de um relógio global para compreender a relação entre os eventos que ocorreram no sistema, sendo uma das contribuições que este trabalho apresenta. Além disso, a proposta resulta em um conjunto de modelos de normalidade que representam um ambiente distribuído para a identificação de anomalias. Também é apresentada uma avaliação do impacto que o tamanho de uma grama tem para a identificação de anomalias, bem como o quão a relação de pertinência sobre os *n*-gramas influencia na classificação dos traços de execuções.

Foram avaliadas duas estratégias de construção de modelos de normalidade (união e interseção). Os modelos construídos foram avaliados em relação à sua capacidade de representar o comportamento normal do sistema e detectar anomalias. Para esta avaliação, foi utilizado uma base contendo execuções reais de uma aplicação distribuída e de um conjunto de ataques verdadeiros executados em contextos diferentes. Os resultados obtidos mostram que os modelos baseados em união oferecem uma boa capacidade em modelar o comportamento normal, demonstrando resultados promissores em termos de acurácia e taxas de verdadeiros positivos. Os modelos baseados em interseção se mostraram mais sensíveis para detectar anomalias, mas não modelam bem a normalidade. Além disso, observa-se que *n*-gramas mais longos detectam ataques mais facilmente, porém são mais propensos a falsos positivos. Dentro do contexto da pesquisa, a análise em torno da relação de pertinência total entre os conjuntos comprovou ser uma abordagem inviável, já que os modelos são incapazes de representar todos os possíveis comportamentos que podem ocorrer dentro de um sistema.

Ao considerar a identificação de anomalias e da normalidade separadamente, o estudo consegue definir resultados claros em relação à aplicação de *n*-gramas, destacando o impacto que a representação do ambiente possui em relação ao tamanho da grama. Visto que quanto maior o tamanho da grama, mais específico é o comportamento aceito, diminuindo a probabilidade de ocorrência entre os traços analisados.

Apesar dos resultados iniciais se mostrarem promissores, alguns questionamentos permanecem em aberto, abrindo caminhos para pesquisas futuras. É visível a necessidade de aumentar o número de execuções normais e de ataques na base de traços, para ter uma representação mais ampla e equilibrada dos comportamentos possíveis. A construção de modelos baseados em interseção apresentou uma restrição significativa, a flexibilização durante a combinação desses subconjuntos é uma oportunidade de contornar essa limitação. Outra abordagem consiste em introduzir a noção de generalização na representação dos *n*-gramas, afim de obter uma representação mais abrangente (e ao mesmo tempo mais compacta) do comportamento normal do sistema.

REFERÊNCIAS

Angiulli, F., Argento, L. e Furfaro, A. (2015). Exploiting *n*-gram location for intrusion detection. Em *Proceedings of the 27th International Conference on Tools with Artificial Intelligence*, páginas 1093–1098, Vietri sul Mare, Itália. IEEE.

Arlot, S., Celisse, A. et al. (2010). A survey of cross-validation procedures for model selection. *Statistics surveys*, 4:40–79.

Barrón-Cedeño, A. e Rosso, P. (2009). On automatic plagiarism detection based on n-grams comparison. Em *European conference on information retrieval*, páginas 696–700. Springer.

Beschastnikh, I., Brun, Y., Ernst, M. D. e Krishnamurthy, A. (2014). Inferring models of concurrent systems from logs of their behavior with CSight. Em *Proceedings of the 36th International Conference on Software Engineering*, páginas 468–479, Hyderabad, Índia. ACM.

Beschastnikh, I., Brun, Y., Ernst, M. D., Krishnamurthy, A. e Anderson, T. E. (2011). Mining temporal invariants from partially ordered logs. Em *Managing Large-scale Systems via the Analysis of System Logs and the Application of Machine Learning Techniques*, páginas 1–10. ACM.

Bhargava, B. e Lilien, L. (2004). Vulnerabilities and threats in distributed systems. Em *International Conference on Distributed Computing and Internet Technology*, páginas 146–157. Springer.

Blostein, D. e Grbavec, A. (1997). Recognition of mathematical notation. Em *Handbook of character recognition and document image analysis*, páginas 557–582. World Scientific.

Catania, C. A. e Garino, C. G. (2012). Automatic network intrusion detection: Current techniques and open issues. *Computers & Electrical Engineering*, 38(5):1062–1072.

Coulouris, G., Dollimore, J., Kindberg, T. e Blair, G. (2013). *Sistemas Distribuídos: Conceitos e Projeto*. Bookman Editora, 5 edition.

Du, M., Li, F., Zheng, G. e Srikumar, V. (2017). Deeplog: Anomaly detection and diagnosis from system logs through deep learning. Em *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, páginas 1285–1298.

Forrest, S., Hofmeyr, S. A., Somayaji, A. e Longstaff, T. A. (1996). A sense of self for Unix processes. Em *Proceedings 1996 IEEE Symposium on Security and Privacy*, páginas 120–128. IEEE.

Fu, Q., Lou, J.-G., Wang, Y. e Li, J. (2009). Execution anomaly detection in distributed systems through unstructured log analysis. Em *Proceedings of the 9th International Conference on Data Mining*, páginas 149–158, Miami, FL, EUA. IEEE.

Garcia-Teodoro, P., Diaz-Verdejo, J., Maciá-Fernández, G. e Vázquez, E. (2009). Anomaly-based network intrusion detection: Techniques, systems and challenges. *Computers & Security*, 28(1-2):18–28.

Hauser, C., Tronel, F., Fidge, C. e Mé, L. (2013). Intrusion detection in distributed systems, an approach based on taint marking. Em *Proceedings of the International Conference on Communications*, páginas 1962–1967, Budapeste, Hungria. IEEE.

Jia, T., Chen, P., Yang, L., Li, Y., Meng, F. e Xu, J. (2017). An approach for anomaly diagnosis based on hybrid graph model with logs for distributed services. Em *2017 IEEE International Conference on Web Services (ICWS)*, páginas 25–32. IEEE.

Jiang, G., Chen, H., Ungureanu, C. e Yoshihira, K. (2006). Multiresolution abnormal trace detection using varied-length *n*-grams and automata. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 37(1):86–97.

Khraisat, A., Gondal, I., Vamplew, P. e Kamruzzaman, J. (2019). Survey of intrusion detection systems: techniques, datasets and challenges. *Cybersecurity*, 2(1):20.

Kolbeck, B., Stender, J., Berlin, M., Kleineweber, C., Noack, Matthias adn Seiferth, P., Langner, F., HPCEurope, N., Hupfeld, F., Gonzales, J., Schäfer, P., Kairies, L., Fischer, J. V., Dillmann, J. e Schmidtke, R. (2015). The XtreemFS installation and user guide. Relatório técnico, XtreemFS.

Kshemkalyani, A. D. e Singhal, M. (2011). *Distributed computing: principles, algorithms, and systems*. Cambridge University Press.

Lamport, L. (1978). Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565.

Lanoë, D., Hurfin, M., Totel, E. e Maziero, C. (2019). An efficient and scalable intrusion detection system on logs of distributed applications. Em *Proceedings of the International Conference on ICT Systems Security and Privacy Protection*, páginas 49–63, Lisboa, Portugal. Springer.

Lee, D. e Yannakakis, M. (1996). Principles and methods of testing finite state machines - a survey. *Proceedings of the IEEE*, 84(8):1090–1123.

Liao, H.-J., Lin, C.-H. R., Lin, Y.-C. e Tung, K.-Y. (2013). Intrusion detection system: A comprehensive review. *Journal of Network and Computer Applications*, 36(1):16–24.

Lorenzoli, D., Mariani, L. e Pezzè, M. (2006). Inferring state-based behavior models. Em *Proceedings of the International Workshop on Dynamic Systems Analysis*, páginas 25–32, Shanghai, China. ACM.

Maslan, A., Mohammad, K. M. e Arnomo, S. A. (2018). DDoS detection on network protocol using cosine similarity and *n*-gram+ method. Em *Proceedings of the International Conference on Sustainable Information Engineering and Technology*, páginas 234–239, Malang, Indonésia. IEEE.

Mattern, F. et al. (1988). Virtual time and global states of distributed systems. Em *Proceedings of the International Workshop on Parallel and Distributed Algorithms*, Chateau de Bonas,France. Elsevier Science Publishers.

Nguyen, X. N., Vu, L. H. et al. (2016). Pocad: a novel pay load-based one-class classifier for anomaly detection. Em *2016 3rd national foundation for science and technology development conference on information and computer science (NICS)*, páginas 74–79. IEEE.

Nisioti, A., Mylonas, A., Yoo, P. D. e Katos, V. (2018). From intrusion detection to attacker attribution: A comprehensive survey of unsupervised methods. *IEEE Communications Surveys & Tutorials*, 20(4):3369–3388.

Perdisci, R., Ariu, D., Fogla, P., Giacinto, G. e Lee, W. (2009). McPAD: A multiple classifier system for accurate payload-based anomaly detection. *Computer Networks*, 53(6):864–881.

Quobyte Inc (2020). XtreemFS - fault-tolerant distributed file system. `http://www.xtreemfs.org`.

Raguenet, I. e Maziero, C. (2008). A fuzzy model for the composition of intrusion detectors. Em *Proceedings of the 23rd International Information Security Conference*, páginas 237–251, Milão, Itália. Springer.

Rasmussen, C. (2014). Gaussian processes in machine learning advanced lectures in machine learning. *Lecture Notes in Computer Science*, 3176:6371.

Santos, I., Penya, Y. K., Devesa, J. e Bringas, P. G. (2009). N-grams-based file signatures for malware detection. *ICEIS (2)*, 9:317–320.

Scarfone, K. e Mell, P. (2012). Guide to intrusion detection and prevention systems (IDPS). Relatório técnico, National Institute of Standards and Technology.

Shana, J. e Venkatachalam, T. (2015). An improved method for counting frequent itemsets using bloom filter. *Procedia Computer Science*, 47:84–91.

Sidorov, G., Velasquez, F., Stamatatos, E., Gelbukh, A. e Chanona-Hernández, L. (2014). Syntactic n-grams as machine learning features for natural language processing. *Expert Systems with Applications*, 41(3):853–860.

Tanenbaum, A. S. e Van Steen, M. (2007). *Sistemas Distribuídos - Princípios e Paradigmas*. Prentice-Hall.

Totel, E., Hkimi, M., Hurfin, M., Leslous, M. e Labiche, Y. (2016). Inferring a distributed application behavior model for anomaly based intrusion detection. Em *Proceedings of the 12th European Dependable Computing Conference*, páginas 53–64, Gotemburgo, Suécia. IEEE.

Vidal, J. M. e Monge, M. A. S. (2019). Adversarial communication networks modeling for intrusion detection strengthened against mimicry. Em *Proceedings of the 14th International Conference on Availability, Reliability and Security*, páginas 1–6, Canterbury, Reino Unido. ACM.

Vidal, J. M., Orozco, A. L. S. e Villalba, L. J. G. (2017). Alert correlation framework for malware detection by anomaly-based packet payload analysis. *Journal of Network and Computer Applications*, 97:11–22.

Viescinski, A., Heinrich, T., Will, N. C. e Maziero, C. (2020). Construção de modelos baseados em n-gramas para detecção de anomalias em aplicações distribuídas. Em *XX Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais*, Petrópolis, RJ, Brazil. SBC.

Villalba, L. J. G., Orozco, A. L. S. e Vidal, J. M. (2017). Advanced payload analyzer preprocessor. *Future Generation Computer Systems*, 76:474–485.

Wang, K. e Stolfo, S. J. (2004). Anomalous payload-based network intrusion detection. Em *Proceedings of the International Workshop on Recent Advances in Intrusion Detection*, páginas 203–222, Sophia Antipolis, França. Springer.

Wressnegger, C., Schwenk, G., Arp, D. e Rieck, K. (2013). A close look on *n*-grams in intrusion detection: anomaly detection vs. classification. Em *Proceedings of the Workshop on Artificial Intelligence and Security*, páginas 67–76, Berlim, Alemanha. ACM.

Zolotukhin, M. e Hämäläinen, T. (2013). Detection of anomalous HTTP requests based on advanced *n*-gram model and clustering techniques. Em *Proceedings of the 13th International Conference on Internet of Things, Smart Spaces, and Next Generation Networking*, páginas 371–382. Springer, São Petersburgo, Rússia.

APÊNDICE A – PUBLICAÇÕES

A.1 ARTIGOS PUBLICADOS NO CONTEXTO DA DISSERTAÇÃO

- **Viescinski, A.** ; Heinrich, T. ; Will, N. C. ; Maziero, C. A.(2020). Construção de modelos baseados em N-gramas para detecção de anomalias em aplicações distribuídas. *XX Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais (SBSeg'2020)*.

A.2 ARTIGOS PUBLICADOS NO CONTEXTO DO GRUPO DE PESQUISA

- Will, N. C.; Heinrich, T. ; **Viescinski, A.** ; Maziero, C. A.(2020). A Trusted Message Bus Built on Top of D-Bus. *XX Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais (SBSeg'2020)*.

- Will, N. C.; Heinrich, T. ; **Viescinski, A.** ; Maziero, C. A.(2021). Trusted Inter-Process Communication Using Hardware Enclaves. *15th Annual IEEE International Systems Conference (SysCon'2021)*.