Universidade Federal de São Carlos– UFSCar
Centro de Ciências Exatas e de Tecnologia– CCET
Departamento de Computação– DC
Programa de Pós-Graduação em Ciência da Computação– PPGCC

Daniel Ettore Storolli França

Método de análise de tempo para correções de vulnerabilidades em distribuições Linux

São Carlos
2025

Daniel Ettore Storolli França

Método de análise de tempo para correções de vulnerabilidades em distribuições Linux

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação do Centro de Ciências Exatas e de Tecnologia da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação.

Área de concentração: Redes de Computadores

Orientador: Prof. Dr. Paulo Matias

São Carlos
2025

França, Daniel Ettore Storolli

Método de análise de tempo para correções de vulnerabilidades em distribuições Linux / Daniel Ettore Storolli França -- 2025.
114f.

Dissertação (Mestrado) - Universidade Federal de São Carlos, campus São Carlos, São Carlos
Orientador (a): Paulo Matias
Banca Examinadora: Paulo Matias, Helio Crestana Guardia, Francisco José Monaco
Bibliografia

1. Segurança da Informação. 2. Linux. 3. Métricas temporais. I. França, Daniel Ettore Storolli. II. Título.

Ficha catalográfica desenvolvida pela Secretaria Geral de Informática (SIn)

DADOS FORNECIDOS PELO AUTOR

Bibliotecário responsável: Arildo Martins - CRB/8 7180

UNIVERSIDADE FEDERAL DE SÃO CARLOS

Centro de Ciências Exatas e de Tecnologia
Programa de Pós-Graduação em Ciência da Computação

Folha de Aprovação

Defesa de Dissertação de Mestrado do candidato Daniel Ettore Storolli França, realizada em 03/12/2025.

Comissão Julgadora:

Prof. Dr. Paulo Matias (UFSCar)

Prof. Dr. Helio Crestana Guardia (UFSCar)

Prof. Dr. Francisco José Monaco (USP)

O Relatório de Defesa assinado pelos membros da Comissão Julgadora encontra-se arquivado junto ao Programa de Pós-Graduação em Ciência da Computação.

Este trabalho é dedicado a toda a comunidade de software livre,
em especial aos que se dedicam ao seu desenvolvimento e divulgação.

Agradecimentos

Em primeiro lugar, gostaria de expressar minha sincera gratidão a Deus e a Nossa Senhora Aparecida pela força espiritual que me acompanhou nesta jornada. À minha família, em especial à minha dedicada esposa Juliana e aos meus amados filhos Eliza e Ettore, agradeço pela paciência incomparável, apoio incondicional e compreensão durante todos os momentos deste desafio acadêmico. Registro aqui também minha eterna gratidão aos queridos amigos João Tenório Neto (in memoriam) e José Valdemir Begnami (in memoriam), cujas memórias continuam a me inspirar.

Dirijo meus profundos agradecimentos ao meu orientador, Dr. Paulo Matias, pela confiança depositada em meu trabalho, pela orientação criteriosa, pelo comprometimento exemplar e pela paciência que dedicou ao longo de todo o desenvolvimento desta pesquisa. Sua expertise e disponibilidade foram fundamentais para a conclusão deste projeto.

Aos ilustres docentes do Departamento de Computação (DC) da Universidade Federal de São Carlos (UFSCar), reconheço com gratidão os conhecimentos compartilhados, as valiosas sugestões e o apoio, em especial aos docentes Dr. Hélio Crestana Guardia e Dr. Sérgio Donizetti Zorzo.

Agradeço igualmente ao professor e pesquisador Dr. Marcos Antônio Simplicio Junior, do Departamento de Engenharia de Computação e Sistemas Digitais da Escola Politécnica da Universidade de São Paulo (USP), pelas pertinentes observações e orientações técnicas oferecidas durante a etapa de qualificação desta dissertação. Reconheço com apreço o apoio fundamental do Dr. Erick Lazaro Melo e de todos os profissionais da Secretária Geral de Informática (SIn) da UFSCar, cujo apoio se manifestou tanto em diálogos enriquecedores quanto em questões técnicas. Agradeço especialmente a Fabiano Losilla de Carvalho, Renato de Souza Gomes e Mesailde de Souza de Oliveira Matias. Por fim, o presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001.

"Qual é a tarefa mais difícil do mundo?
Pensar"
(Ralph Waldo Emerson)

Resumo

Sistemas operacionais baseados em distribuições Linux são amplamente utilizados, mas, assim como qualquer software, estão sujeitos a vulnerabilidades de segurança que podem causar interrupções críticas de serviços ou acesso não autorizado a dados sensíveis. Essas falhas, formalmente catalogadas como *Common Vulnerabilities and Exposures (CVE)*, após identificação por uma *CVE Numbering Authority (CNA)*, são registradas com um *Common Vulnerabilities and Exposures Identifier (CVE ID)*. A divulgação pública imediata desses registros nem sempre ocorre, pois a exposição prematura pode ser explorada por agentes maliciosos antes da disponibilização de correções, tornando a agilidade na resolução essencial. Este trabalho propõe e aplica o método Coleta e Análise do Tempo de Correção (ColATeC) para investigar o ciclo de vida de vulnerabilidades em distribuições Linux. Foram analisadas as distribuições Red Hat (6, 7, 8 e 9), AlmaLinux (8 e 9), Rocky Linux (8 e 9), Debian (10, 11, 12, Sid e Trixie) e Ubuntu (16.04, 18.04, 20.04 e 22.04), com os seguintes objetivos: (i) mapear os mecanismos de divulgação de vulnerabilidades adotados por cada distribuição; (ii) comparar os dados públicos com os registros do *National Institute of Standards and Technology (NIST)*; (iii) quantificar vulnerabilidades por distribuição; (iv) mensurar tempos médios de correção; e (v) avaliar políticas de divulgação entre versões de uma mesma distribuição. Os resultados revelaram variações significativas na eficiência de correção entre as distribuições, identificando aquelas com maior densidade de vulnerabilidades e respostas mais ágeis. Além disso, detectaram-se discrepâncias entre os dados publicados, indicando oportunidades de aprimoramento na transparência e na adoção de formatos estruturados (ex. *JavaScript Object Notation (JSON)*). Conclui-se que o método ColATeC configura-se como ferramenta estratégica para monitoramento contínuo de métricas de segurança cibernética. Sua adoção visa subsidiar a tomada de decisões e contribuir para o fortalecimento da postura de segurança em ecossistemas baseados em Linux e outros softwares.

Palavras-chave: Segurança da Informação. Linux. CVE. Métricas temporais.

Abstract

Linux-based operating systems are widely used; however, like any software, they are susceptible to security vulnerabilities that can lead to critical service disruptions or unauthorized access to sensitive data. These flaws, formally cataloged as Common Vulnerabilities and Exposures (CVE) after identification by a CVE Numbering Authority (CNA), are registered with a Common Vulnerabilities and Exposures ID (CVE ID). Immediate public disclosure of these records does not always occur, as premature exposure can be exploited by malicious actors before patches are available, making swift resolution essential. This work proposes and applies the Collection and Analysis of Correction Time method (ColATeC) to investigate the lifecycle of vulnerabilities in Linux distributions. The analyzed distributions include Red Hat (6, 7, 8, and 9), AlmaLinux (8 and 9), Rocky Linux (8 and 9), Debian (10, 11, 12, Sid, and Trixie), and Ubuntu (16.04, 18.04, 20.04, and 22.04), with the following objectives: (i) map vulnerability disclosure mechanisms adopted by each distribution; (ii) compare public data with records from the National Institute of Standards and Technology (NIST); (iii) quantify vulnerabilities per distribution; (iv) measure average patching times; and (v) evaluate disclosure policies across versions of the same distribution. The results revealed significant variations in patching efficiency among distributions, identifying those with higher vulnerability density and faster response times. Additionally, discrepancies in published data were detected, suggesting opportunities for improvement in transparency and the adoption of structured formats (e.g. JavaScript Object Notation (JSON)). In conclusion, the ColATeC method establishes itself as a strategic tool for continuous monitoring of cybersecurity metrics. Its adoption aims to support decision-making and strengthen security postures in Linux-based ecosystems and other software environments.

Keywords: Information Security. Linux. CVE. Temporal Metrics.

Lista de ilustrações

Figura 1 – Diagrama simplificado do fluxo de processamento do método ColATeC 35
Figura 2 – Exemplo de arquivo JSON contendo dados do método ColATeC 36
Figura 3 – Trecho de arquivo JSON fornecido por Red Hat 37
Figura 4 – Diagrama simplificado do fluxo completo do método ColATeC 40
Figura 5 – Exemplo dos registros no banco de dados 41
Figura 6 – Exemplo dos registros no banco de dados - padronização dos nomes de pacotes 42
Figura 7 – Opções de filtros de uma busca avançada do NIST 51
Figura 8 – Exemplo *changelog* Debian 56
Figura 9 – Exemplo de vulnerabilidade no Debian Buster com versões conflitantes 57
Figura 10 – Informação sobre as versões dos pacotes Ubuntu e Ubuntu Pro 59
Figura 11 – Comparação CVE-2021-3928 entre Ubuntu e Ubuntu PRO 59
Figura 12 – Total de CVEs Únicas no Período de 5 Anos 65
Figura 13 – Total de CVEs Únicas Ano a Ano - Geral 66
Figura 14 – Total de CVEs Únicas Ano a Ano - Pacotes Comuns 66
Figura 15 – Comparação de Correções AlmaLinux e Rocky Linux vs. Red Hat . . . 68
Figura 16 – Histograma de Diferença de Dias: AlmaLinux vs. Red Hat 69
Figura 17 – Histograma de Diferença de Dias: Rocky Linux vs. Red Hat 70
Figura 18 – Comparação de Severidades de CVEs Corrigidas Dia Zero - Família .deb 76
Figura 19 – Comparação de Severidades de CVEs Corrigidas Maior que Zero Dias - Família .deb 77
Figura 20 – Comparação de Severidades de CVEs Corrigidas Dia Zero - Família .rpm 78
Figura 21 – Comparação de Severidades de CVEs Corrigidas Maior que Zero Dias - Família .rpm 79
Figura 22 – Comparação de Severidades de CVEs Não Corrigidas - Família .deb . . 81
Figura 23 – Comparação de Severidades de CVEs Com Datas Negativas - Família .deb 82

Figura 24 – Comparação de Severidades de CVEs Não Corrigidas - Família .rpm . . 83
Figura 25 – Comparação de Severidades de CVEs Com Datas Negativas - Família .rpm 84
Figura 26 – Detalhamento de CVEs Não Corrigidos - Red Hat 86
Figura 27 – Detalhamento de CVEs Não Corrigidos - Ubuntu 87
Figura 28 – Corrigidas apenas no Ubuntu Pro - Geral 89
Figura 29 – Tempo de Correção Geral - 5 Anos 91
Figura 30 – Tempo de Correção Pacotes em Comum - 5 Anos 92
Figura 31 – Tempo de Correção Geral - Ano a Ano 94
Figura 32 – Tempo de Correção Pacotes em Comum - Ano a Ano 95
Figura 33 – Comparação Agregada - Geral 97
Figura 34 – Comparação Agregada - Pacotes em Comum 98
Figura 35 – Comparação Agregada Versões - Geral 99
Figura 36 – Comparação Agregada Versões - Pacotes em Comum 100

Lista de tabelas

Tabela 1 – Comparativo dos trabalhos relacionados com o trabalho proposto . . . 32
Tabela 2 – Resumo entidade-relacionamento do banco **cvedb5** 43
Tabela 3 – Dados comparativos da CVE-2023-21866 44
Tabela 4 – Porcentagem de Correções não Calculadas no Debian 58
Tabela 5 – Porcentagem de Correções não Calculadas no Ubuntu 61
Tabela 6 – Tabela Geral de Distribuição de Correções 71
Tabela 7 – Tabela de Distribuição de Correções - Pacotes Comuns 71
Tabela 8 – Comparações de severidade entre as distribuições e o NIST - Geral 73
Tabela 9 – Comparações de severidade entre as distribuições e o NIST - Pacotes em Comum . 74
Tabela 10 – CVE com severidades diferentes na mesma distribuição 75

Lista de siglas

ABNT Associação Brasileira de Normas Técnicas

ALBA AlmaLinux Bug Advisory

ALEA AlmaLinux Enhancement Advisory

ALSA AlmaLinux Security Advisory

API Application Programming Interface

CAPES Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil

CERT.br Centro de Estudos, Resposta e Tratamento de Incidentes de Segurança no Brasil

CNA CVE Numbering Authority

ColATeC Coleta e Análise do Tempo de Correção

CSV Comma-Separated Values

CVD Coordinated Vulnerability Disclosure

CVE Common Vulnerabilities and Exposures

CVE ID Common Vulnerabilities and Exposures Identifier

CVSS Common Vulnerability Scoring System

DC Departamento de Computação

DSA Debian Security Advisor

DIFFCVSS OS-aware vulnerability prioritization

ETL Extract, Transform, Load

FIRST Forum of Incident Response and Security Teams

HTTP Hypertext Transfer Protocol

IA Inteligência Artificial

IEC International Electrotechnical Commission

IoT Internet of Things

IRC Internet Relay Chat

ISO International Organization for Standardization

JSON JavaScript Object Notation

LLM Large Language Models

NIC.br Núcleo de Informação e Coordenação do Ponto BR

NIST National Institute of Standards and Technology

NVD National Vulnerability Database

PyPI Python Package Index

RHBA Red Hat Bug Advisory

RHEA Red Hat Enhancement Advisory

RHEL Red Hat Enterprise Linux

RHSA Red Hat Security Advisory

RLSA Rocky Linux Security Advisories

RLBA Rocky Linux Bugfix Advisories

RLEA Rocky Linux Enhancement Advisories

SIEM Security Information and Event Management

SIn Secretária Geral de Informática

SQL Structured Query Language

TI Tecnologia da Informação

UFSCar Universidade Federal de São Carlos

URL Uniform Resource Locator

USN Ubuntu Security Notices

USP Universidade de São Paulo

UTC Coordinated Universal Time

VM Virtual Machine

XML Extensible Markup Language

Sumário

1 INTRODUÇÃO 23
1.1 Motivação e objetivos 24

2 REVISÃO DA LITERATURA 27
2.1 Trabalhos relacionados 27
2.2 Comparação com este trabalho 34
2.2.1 Organização do método ColATeC 35

3 METODOLOGIA 39
3.1 Uso de inteligência artificial (Inteligência Artificial (IA)) 39
3.2 O método ColATeC 40
3.2.1 Arquitetura e fluxo de execução 40
3.3 Definição das métricas 46
3.3.1 Tempo de correção de vulnerabilidade 46
3.4 Delineamento experimental 47
3.4.1 Seleção das distribuições analisadas 47
3.4.2 Fontes de dados 47
3.4.3 Período de coleta e análise 48
3.5 Ameaças à validade 49
3.5.1 Ameaças à validade de construção 49
3.5.2 Ameaças à validade interna 49
3.5.3 Ameaças à validade externa 49
3.5.4 Limitações das fontes de dados 49

4 ESTUDO DE CASO - COLETA DOS DADOS 51
4.1 NIST 51
4.2 Linux Red Hat 52

4.3 AlmaLinux . . . 53
4.4 Rocky Linux . . . 54
4.5 Debian . . . 55
4.6 Ubuntu . . . 58

5 RESULTADOS E ANÁLISE DO ESTUDO DE CASO 63
5.1 Vulnerabilidades de pacotes comuns entre ecossistemas . . . 63
5.2 Apresentação geral dos dados . . . 64
5.2.1 Análise mais detalhada sobre AlmaLinux e Rocky Linux . . . 67
5.3 Estado de correção das vulnerabilidades . . . 70
5.3.1 Criticidade das vulnerabilidades . . . 73
5.4 Vulnerabilidades não corrigidas e com datas negativas . . . 80
5.4.1 Criticidade das vulnerabilidades não corrigidas e com datas negativas . 80
5.5 Análise detalhada do Red Hat e Ubuntu - vulnerabilidades não corrigidas . . . 85
5.6 Análise da criticidade de vulnerabilidades corrigidas apenas no Ubuntu Pro . . . 88
5.7 Apresentação do tempo médio de correção . . . 90

6 CONCLUSÃO . . . 103
6.1 Trabalhos Futuros . . . 105

REFERÊNCIAS . . . 109

Capítulo 1

Introdução

Segundo SCHWAB (1), a aceleração tecnológica característica da Quarta Revolução Industrial consolidou a dependência estratégica de sistemas computacionais em todos os setores da sociedade, desde instituições acadêmicas até organizações públicas e privadas. Nesse contexto, a infraestrutura de Tecnologia da Informação (TI) tornou-se um elemento crítico para operações essenciais, exigindo mecanismos robustos de proteção cibernética.

As distribuições Linux constituem a base de grande parte da infraestrutura global de TI, abrangendo desde servidores corporativos até sistemas embarcados e ambientes de desenvolvimento. A confiança depositada nessas distribuições está diretamente vinculada à capacidade de suas comunidades em identificar e corrigir vulnerabilidades de segurança de forma ágil. No entanto, assim como as soluções proprietárias, esses sistemas permanecem suscetíveis a explorações maliciosas que podem comprometer a disponibilidade de serviços e a confidencialidade de dados.

Dados do Centro de Estudos, Resposta e Tratamento de Incidentes de Segurança no Brasil (CERT.br) (2) revelam a dimensão dos incidentes cibernéticos registrados em redes monitoradas pelo Núcleo de Informação e Coordenação do Ponto BR (NIC.br) (3). Tais informações corroboram a necessidade de uma abordagem de proteção universal para ecossistemas digitais, que transcenda especificidades de fabricantes, sistemas operacionais ou particularidades técnicas.

Segundo o NIST (4), uma vulnerabilidade de segurança pode ser definida como uma falha ou fragilidade no código ou no *design* de um sistema, passível de exploração por agentes maliciosos para comprometer a confidencialidade, integridade ou disponibilidade das informações.

O período entre a descoberta de uma falha e sua efetiva mitigação constitui um intervalo crítico, denominado “janela de vulnerabilidade”, que determina diretamente o risco

de exploração. Um estudo pioneiro de ARBAUGH, W. et al. (5) constatou que invasões em massa são exploradas não apenas pela divulgação da falha, mas pela automação de sua exploração, tornando a agilidade na aplicação de correções um fator decisivo para a segurança.

Em linhas gerais, a aplicação do ColATeC utilizando os códigos desenvolvidos para este trabalho indica que Ubuntu e Red Hat apresentam os menores tempos medianos de correção, enquanto AlmaLinux e Rocky Linux exibem lacunas de catalogação, e o Debian sofre com a maior variabilidade temporal. Esses achados reforçam que a resposta a vulnerabilidades varia significativamente entre os ecossistemas estudados e motivam a estruturação metodológica proposta ao longo deste texto.

Quando identificadas por uma CNA, as vulnerabilidades são catalogadas e recebem identificadores únicos do tipo CVE ID. A divulgação pública desses registros nem sempre é imediata, pois a exposição prematura de uma falha permite que agentes mal-intencionados a explorem antes que as correções sejam implementadas pelos mantenedores. Diante desse cenário, a agilidade na correção de segurança configura-se como um fator crítico para a proteção de sistemas.

1.1 Motivação e objetivos

De acordo com CIPHER (6), a análise e a correção de vulnerabilidades conhecidas são mecanismos essenciais para a proteção de ativos digitais. Essas falhas representam vetores de ataque para agentes mal-intencionados, configurando portas de entrada para ações que variam desde interrupções operacionais até violações de dados. Corroborando essa perspectiva, um relatório da IBM (7) identifica a exploração de vulnerabilidades como o segundo vetor de ataque mais prevalente.

A pesquisa em segurança de *software* frequentemente parte da premissa de que descobrir e corrigir vulnerabilidades é uma certeza. Contudo, essa visão foi desafiada por RESCORLA (8), que argumentou, sob uma ótica econômica, que o ato de encontrar falhas poderia diminuir a segurança do ecossistema. Seu argumento central sustenta que a informação sobre novas vulnerabilidades se dissemina para atacantes mais rapidamente do que as correções são aplicadas pelos usuários, criando um período de risco elevado que antes não existia. Esse efeito resulta de um profundo desalinhamento de incentivos entre fornecedores, pesquisadores e usuários.

Tal perspectiva crítica ressalta que as soluções para a insegurança de *software* não são apenas técnicas, mas também envolvem dimensões econômicas. Isso justifica a importância de estudos que, como este, se propõem a medir e otimizar uma das fases mais críticas do processo: o tempo para correção. A medição desse tempo consolida-se, portanto, como uma métrica estratégica para a redução de superfícies de ataque e a gestão proativa de riscos.

Por esses motivos, este trabalho propõe um método para coletar, analisar e comparar o tempo de correção de vulnerabilidades em diferentes distribuições Linux. O objetivo principal é desenvolver uma abordagem sistemática, denominada ColATeC (Coleta e Análise do Tempo de Correção), que permita extrair dados de fontes públicas, processá-los e apresentar uma análise quantitativa do ciclo de vida das correções de segurança.

Os objetivos específicos do método são:

i) **Estabelecer padrões para a publicação de vulnerabilidades e correções**: A padronização de dados entre comunidades de *software* livre e outros repositórios (como MacPorts e Homebrew no macOS e o Chocolatey no Windows) simplificaria a coleta de informações de forma eficiente;

ii) **Viabilizar a coleta automatizada em larga escala**: Implementar essa capacidade permitiria a criação de bases históricas para análises temporais de segurança;

iii) **Possibilitar a escolha fundamentada de distribuições**: Ser um dos critérios que permita ajudar usuários finais a escolherem sistemas operacionais com base em evidências quantitativas de segurança e eficiência conforme suas necessidades;

iv) **Fomentar melhores práticas de desenvolvimento**: A aplicação do ColATeC pode evidenciar disparidades de desempenho entre projetos, incentivando equipes a otimizar seus processos de correção.

Esses objetivos específicos conectam-se diretamente às questões de pesquisa delineadas na sequência. A padronização de dados (objetivo *i*) fundamenta a identificação das informações indispensáveis ao método (Q1) e prepara o terreno para comparações entre ecossistemas. A coleta automatizada (objetivo *ii*) viabiliza examinar como cada distribuição divulga vulnerabilidades e corrige falhas (Q2 e Q4). A possibilidade de escolha fundamentada de distribuições (objetivo *iii*) responde às questões sobre volume de falhas e tempos médios (Q3 e Q4). Por fim, o fomento a melhores práticas (objetivo *iv*) cria evidências para discutir diferenciações entre versões e políticas de manutenção (Q5).

Para avaliar a aplicabilidade do método, foi conduzido um estudo de caso envolvendo as distribuições Linux Red Hat, AlmaLinux, Rocky Linux, Debian e Ubuntu. O recorte privilegia sistemas amplamente adotados em ambientes corporativos e comunitários, incluindo derivadas compatíveis com o Red Hat Enterprise Linux (RHEL), o que permite comparar processos de manutenção sob premissas semelhantes de suporte. Modelos *rolling release*, como openSUSE Tumbleweed ou Arch Linux, foram considerados para trabalhos futuros, pois exigiriam adaptações metodológicas em razão do fluxo contínuo de atualizações. A análise se concentrou em dois eixos: (i) os mecanismos de divulgação de vulnerabilidades e (ii) a eficiência temporal nos processos de correção. A metodologia busca responder às seguintes questões de pesquisa:

Q1 - Quais informações são essenciais para a aplicação do método ColATeC?

Q2 - Como cada distribuição divulga suas vulnerabilidades de segurança?

Q3 - Qual distribuição apresenta o maior número de falhas de segurança reportadas?

Q4 - Qual distribuição possui o menor tempo médio de correção de vulnerabilidades?

Q5 - Existe tratamento diferenciado para versões distintas de uma mesma distribuição?

A contribuição deste trabalho consiste na criação de um método replicável e na disponibilização de uma análise comparativa que pode auxiliar administradores de sistemas, desenvolvedores e pesquisadores na tomada de decisões mais embasadas sobre a seleção e gestão de distribuições Linux. Para alcançar os objetivos propostos, este documento está estruturado da seguinte forma:

1. **Capítulo 2: Revisão da literatura**: Apresenta estudos que analisaram o ciclo de vida e o tempo de correção de vulnerabilidades, destacando metodologias e lacunas que este trabalho busca preencher;

2. **Capítulo 3: Metodologia**: Descreve em detalhes o método ColATeC e suas fases, justificando as escolhas tecnológicas e as fontes de dados;

3. **Capítulo 4: Estudo de caso - coleta dos dados**: Expõe os dados coletados e processados pela aplicação do ColATeC;

4. **Capítulo 5: Resultados e análise do estudo de caso**: Apresenta a análise comparativa detalhada do tempo de correção entre as distribuições;

5. **Capítulo 6: Conclusão**: Resume os resultados, discute as contribuições do trabalho, aponta suas limitações e sugere direções para pesquisas futuras.

Todo o código-fonte desenvolvido neste trabalho está disponível em <https://github.com/daniel-franca/mestrado_ppgcc>. Para cada código de coleta de dados, foi necessário realizar um mapeamento manual das informações para seu armazenamento em tabelas do banco de dados, devido à ausência de padronização na forma como esses dados são disponibilizados pelas fontes. Cada mapeamento teve como objetivo identificar os campos exigidos pelo método ColATeC, extraindo as informações. Apesar desse esforço, algumas distribuições ainda ocultam parte das informações do ciclo de vida ou mantêm campos relevantes apenas em formatos semi-estruturados. Nessas situações, o método recorre a *parsers* dedicados (por exemplo, para extrair dados de *changelogs* do Debian e do Ubuntu) e registra a ausência dos campos quando não há fonte legítima. Essa limitação – analisada em detalhe nos capítulos subsequentes – reforça a necessidade de iniciativas comunitárias para aumentar a transparência e a interoperabilidade dos repositórios de segurança.

Capítulo 2

Revisão da literatura

A análise do ciclo de vida de vulnerabilidades é um campo consolidado na pesquisa em segurança de *software*. Diversos estudos buscaram compreender as dinâmicas entre a descoberta, a divulgação, a exploração e a correção de falhas. Este capítulo apresenta uma revisão crítica dos trabalhos mais relevantes, contextualizando a contribuição do presente estudo.

A gestão de vulnerabilidades constitui um desafio crítico de segurança cibernética. Um conceito fundamental é a janela de vulnerabilidade (período entre a descoberta de uma falha e sua correção). Conforme definido por ARBAUGH, W. et al. (5), tal janela representa um intervalo de risco crítico. A literatura evidencia lacunas metodológicas na análise comparativa de vulnerabilidades, particularmente na padronização de métricas temporais e coleta automatizada de dados. Este capítulo examina trabalhos correlatos e fundamenta a proposta do método ColATeC.

Esta revisão assume natureza narrativa: o corpus contempla estudos clássicos e contemporâneos apontados pela literatura especializada e por revisores externos do projeto, privilegiando referências que iluminam lacunas de padronização, automação e comparabilidade com distribuições Linux. Não foi conduzido protocolo sistemático formal (por exemplo, PRISMA), e essa decisão é explicitada para delimitar o escopo interpretativo das sínteses apresentadas.

2.1 Trabalhos relacionados

JANSON (9) investigou vulnerabilidades nas distribuições Ubuntu, Debian, Fedora, Red Hat e OpenSUSE entre 2002-2018, utilizando arquivos *Extensible Markup Language (XML)* do *National Vulnerability Database (NVD)*. Sua metodologia restringiu-se

à análise de *changelogs* locais, apresentando três limitações críticas: (i) seleção de pacotes homogêneos (excluindo pacotes com nomenclaturas divergentes); (ii) exclusão do CentOS devido à ausência de registros no NVD; (iii) baixa significância estatística nos resultados, exceto para OpenSUSE. Essas limitações destacam a necessidade de métodos adaptativos para ecossistemas heterogêneos e fontes de dados alternativas.

PIANTADOSI et al. (10) conduziram um estudo sobre a dinâmica de correção de vulnerabilidades em *software* de código aberto, analisando três dimensões críticas: (i) agentes responsáveis pela correção; (ii) temporalidade do processo; (iii) estratégias técnicas. Os autores analisaram os projetos Apache *Hypertext Transfer Protocol (HTTP) Server* e Apache Tomcat, mapeando manualmente registros de CVEs via mineração de *commits*. Os resultados indicaram que a maioria das vulnerabilidades foi resolvida antes da publicação do CVE, embora algumas correções tenham demorado anos. Contudo, a dependência de procedimentos manuais limita a replicabilidade do estudo em larga escala.

ALFADEL et al. (11) analisaram 550 vulnerabilidades no ecossistema *Python Package Index (PyPI)*, examinando padrões temporais e a eficácia das mitigações. A metodologia integrou dados da plataforma Snyk.io com metadados de pacotes do repositório Libraries.io. Os resultados revelaram um aumento progressivo de vulnerabilidades ao longo do tempo e discrepâncias significativas nas datas de descoberta. Mais de 50% das vulnerabilidades foram corrigidas apenas após sua divulgação pública, indicando uma cultura de manutenção reativa no ecossistema PyPI.

É importante destacar que a gestão de vulnerabilidades no PyPI segue um modelo descentralizado, no qual os repositórios são gerenciados pelos próprios autores dos pacotes. Esses mantenedores são responsáveis por definir *flags* de *status* (como "corrigido" ou "vulnerável") e pela implementação das correções. Conforme observado em ambientes como o Docker Hub, embora os pacotes sejam disponibilizados publicamente, sua manutenção depende diretamente da ação proativa dos autores. Esse modelo de gestão baseado no usuário, embora promova autonomia, pode resultar em inconsistências no tratamento de vulnerabilidades, especialmente quando os mantenedores não atualizam regularmente os *status* de segurança ou dependem da divulgação pública para iniciar correções. Como limitação metodológica, a dependência de extração manual de dados restringe a replicabilidade do estudo em larga escala. Além disso, as métricas temporais reportadas pelos autores não foram normalizadas para condições equivalentes às das distribuições Linux; os prazos refletem o esforço voluntário de cada mantenedor de pacote, diferindo da responsabilização centralizada típica das distribuições, e, portanto, servem aqui como referências conceituais — não como séries diretamente comparáveis aos resultados obtidos pelo ColATeC.

LIN et al. (12) conduziram uma análise comparativa das distribuições Debian e Fedora, investigando 21.752 e 17.434 vulnerabilidades reportadas, respectivamente. Os dados foram coletados por meio de fontes primárias, incluindo avisos de segurança, repositórios

de pacotes e listas de discussão. Os resultados indicaram que ambas as distribuições mitigaram aproximadamente 50% das vulnerabilidades dentro de 7 dias após a divulgação do CVE. No entanto, os administradores do Fedora gastam mais tempo para testar e disponibilizar correções, resultando em um período de vulnerabilidade mais longo para seus usuários. É relevante notar que o estudo não compara versões distintas de uma mesma distribuição, focando apenas na comparação entre distribuições diferentes.

WU et al. (13) propuseram uma crítica metodológica ao uso convencional do *Common Vulnerability Scoring System (CVSS)* para priorização de vulnerabilidades em ecossistemas Linux heterogêneos. Seu estudo apresenta que a aplicação direta do CVSS tende a superestimar a gravidade das vulnerabilidades devido à diversidade de versões, *forks* e personalizações típicas de distribuições Linux. Como alternativa, os autores desenvolveram *OS-aware vulnerability prioritization (DIFFCVSS)*, um *framework* analítico que pondera o alvo Linux envolvido, resultando em diferentes níveis de criticidade para a mesma CVE em várias situações. O objetivo do DIFFCVSS é determinar se uma vulnerabilidade apresenta maior ou menor severidade em uma versão Linux alternativa em relação à versão principal (ex.: vulnerabilidade classificada como crítica (9,8) no Ubuntu 22.04 pode ser moderada (5,4) no Alpine Linux 3.16). O DIFFCVSS introduz um paradigma de avaliação relativa de riscos, essencial para ecossistemas com alta fragmentação de versões. Seus resultados sugerem a necessidade de revisão dos padrões atuais de classificação de vulnerabilidades para ambientes Linux não homogêneos, reforçando a necessidade de métricas temporais adaptativas, como as propostas pelo ColATeC.

RESCORLA (8) problematiza o impacto econômico da divulgação de vulnerabilidades. Seu modelo revela que a divulgação prematura, sem correções imediatas, amplia a janela de risco, um paradoxo em que a busca por falhas pode, em curto prazo, reduzir a segurança global. Este trabalho adota a métrica temporal para mitigar esse risco, assegurando que a divulgação seja acompanhada de agilidade na correção.

De forma complementar, um estudo fundamental de ARBAUGH, W. et al. (5) propôs um modelo de ciclo de vida para vulnerabilidades (nascimento, descoberta, divulgação, correção, publicidade, automação e morte) e analisou o comportamento de ataques ao longo do tempo. A principal conclusão do trabalho foi que a maioria das invasões ocorria meses ou até anos após a disponibilização das correções pelos desenvolvedores. Os autores identificaram que o verdadeiro catalisador para explorações em larga escala não era a simples divulgação da vulnerabilidade, mas a criação de *scripts* que automatizavam o ataque, permitindo que indivíduos com pouca habilidade técnica (os chamados *script kiddies*) comprometessem sistemas massivamente. Essa constatação sublinha a criticidade da gestão de *patches* e a morosidade dos administradores de sistemas como o principal fator que mantém a janela de vulnerabilidade aberta.

Avançando na quantificação do problema, o estudo de FREI et al. (14) realizou uma análise em larga escala com mais de 80.000 alertas de segurança para medir o ciclo de

vida da vulnerabilidade. Os autores introduziram métricas como o tempo para correção e o tempo para exploração. A principal e mais alarmante conclusão foi a existência de uma lacuna de tempo negativa: na mediana dos casos, um código de exploração (*exploit*) tornava-se público 15 dias antes da disponibilização de uma correção oficial pelo fornecedor. Adicionalmente, os autores destacaram o aumento expressivo de *zero-day exploits* (*exploits* disponíveis no dia da divulgação) e modelaram estatisticamente os tempos de resposta (distribuição de pareto para *exploits*, *Weibull* para *patches*). Este achado revelou que, além da demora na aplicação dos *patches* pelos usuários (conforme ARBAUGH, W. et al. (5)), existe um problema sistêmico na própria indústria, onde os atacantes frequentemente estão um passo à frente dos desenvolvedores. O estudo concluiu que uma estratégia de segurança puramente reativa é insuficiente para proteger os sistemas.

Em um estudo de larga escala baseado em dados de telemetria de mais de 6 milhões de equipamentos, NAYAK et al. (15) apresentaram que a grande maioria das vulnerabilidades divulgadas nunca é explorada no mundo real. Os autores introduziram métricas pragmáticas como a taxa de exploração e a superfície de ataque exercida, concluindo que menos de 35% das falhas nos produtos analisados eram de fato alvo de ataques. Esta descoberta sugere que, embora a medição do tempo para correção seja crucial, uma abordagem de segurança ainda mais eficaz poderia envolver a priorização de *patches* com base na probabilidade de exploração de cada falha específica, focando esforços nas vulnerabilidades que importam na prática. Este trabalho reforça a relevância de métricas temporais adaptativas, como as do ColATeC, para otimizar a alocação de recursos de segurança.

Outra dimensão da análise de vulnerabilidades foi explorada por OZMENT e SCHECHTER (16), que investigaram se a segurança de um *software* melhora com o tempo por meio de um estudo de caso no OpenBSD. A pesquisa introduziu o conceito de vulnerabilidades fundacionais, apresentando que a grande maioria das falhas (62%) e do próprio código (61%) em sistemas maduros consiste em legado. Como principal conclusão, os autores afirmam que, embora os esforços contínuos de segurança sejam eficazes e a taxa de descoberta de falhas antigas diminua, o processo avança de forma extremamente lenta, com uma vida média das vulnerabilidades de pelo menos 2,6 anos.

O estudo ressalta que o gerenciamento de *patches* não representa apenas uma corrida contra atacantes, mas também uma batalha de longo prazo contra a complexidade e o legado da base de código. Esta perspectiva evidencia uma limitação deste trabalho, que se concentra especificamente na questão da gestão de correções. Ressalta-se, contudo, que a pesquisa original foca em uma única distribuição (OpenBSD) e não aborda discrepâncias na divulgação de vulnerabilidades entre diferentes versões.

EDWARDS e CHEN (17) realizaram um estudo longitudinal das versões de projetos de código aberto (Sendmail, Postfix, Apache HTTP *Server* e OpenSSL) combinando análise estática de código-fonte com dados históricos de vulnerabilidades (CVE). Utili-

zando a ferramenta HP *Fortify Source Code Analyzer*, os autores apresentaram as seguintes informações: (i) alterações nas métricas estáticas (ex.: densidade de *issues* críticas) correlacionam-se moderadamente com mudanças na taxa de CVEs/ano; (ii) valores absolutos de métricas (ex.: total de *issues*) **não são comparáveis** entre projetos distintos devido a diferenças intrínsecas de *design*; (iii) a taxa de descoberta de vulnerabilidades tende a diminuir após 3-5 anos do lançamento inicial, indicando maturidade do código. Este trabalho reforça a viabilidade da análise automatizada como indicador de risco, porém ressalta limitações na comparação cruzada de *softwares*, lacuna que o método ColATeC aborda mediante padronização de métricas temporais.

Por fim, BERES et al. (18) propuseram uma abordagem inovadora baseada em modelagem matemática e simulação para que grandes organizações possam avaliar o impacto de suas políticas de segurança. Em vez de apenas analisar dados históricos, a metodologia permite simular cenários e analisar os *trade-off* entre diferentes estratégias, como acelerar a implantação de *patches* versus investir em tecnologias de mitigação pré-*patch*, como sistemas de prevenção de intrusão. Ao construir um modelo que considera tanto o ambiente de ameaças externas quanto os processos internos de uma organização, o trabalho oferece uma ferramenta prática para justificar investimentos e otimizar a gestão de vulnerabilidades. Enquanto o modelo de simulação de BERES et al. (18) foca em políticas de segurança corporativa, o ColATeC propõe um método padronizado para coleta de dados em distribuições Linux, resolvendo lacunas de transparência e automação identificadas na literatura.

A tabela 1 apresenta uma análise comparativa sistematizada entre a literatura e o presente trabalho, destacando aspectos metodológicos, recortes analíticos e contribuições empíricas.

Tabela 1 – Comparativo dos trabalhos relacionados com o trabalho proposto.

Autor	**Foco**	**Compara Versões**	**Fonte**	**Busca**	**Tipo de Verificação**	**Lacuna Relevante ao ColATeC**
JANSON (9)	Distribuições Linux	Não	NVD	Nome do pacote	Análise de *changelog* via *script*	Normalização manual de nomenclaturas; não detalha processo replicável
PIANTADOSI et al. (10)	Apache HTTP *Server* e Tomcat	Não	CVE *Details*	CVE ID	Mineração manual de *commits*	Processo manual sem escalabilidade, restrito a dois projetos
ALFADEL et al. (11)	Ecossistema Python (PyPI)	Não	Snyk.io, Libraries.io	Nome do pacote	Extração manual de dados	Métricas refletem voluntarismo dos mantenedores; dados não padronizados
LIN et al. (12)	Debian e Fedora	Não	Avisos de segurança, repositórios, listas de e-mail	Nome do pacote	Extração de dados via *script*	Compara apenas duas distribuições; não oferece método reutilizável
FREI et al. (14)	Análise em larga escala do ciclo de vida	Não	Alertas de segurança	Análise de dados em massa	Modelagem estatística (*exploits vs. patches)*	Trabalha com dados agregados sem distinguir ecossistemas ou versões
NAYAK et al. (15)	Taxa de exploração de vulnerabilidades	Não	Dados de telemetria de dispositivos	Análise de telemetria	Medição de ataques no mundo real	Foca em exploração, não acompanha tempos de correção ou publicação
EDWARDS e CHEN (17)	Projetos de código aberto (longitudinal)	Sim (do mesmo projeto)	Código-fonte, CVEs	Análise estática de código	Correlação de métricas de código	Métricas absolutas não comparáveis entre projetos distintos
Este trabalho	**Distribuições Linux**	**Sim**	**NVD e dados das próprias distribuições**	**CVE ID**	**Extração automatizada via *script***	**Padroniza campos temporais e preserva severidades específicas por distribuição**

Fonte: Produzido pelo autor

A justaposição dos estudos de JANSON (9) e LIN et al. (12) é particularmente elucidativa para o presente trabalho. JANSON (9) estabelece a viabilidade e a importância de medir empiricamente o tempo de correção de vulnerabilidades em diferentes ecossistemas Linux. Contudo, sua análise permanece no nível dos resultados observados. LIN et al. (12), por sua vez, apresentam a complexidade dos fatores que influenciam esses resultados, mas limitam sua análise aprofundada a apenas duas distribuições. Fica evidente, portanto, a existência de uma lacuna na literatura: a ausência de um estudo que combine a amplitude da análise de JANSON (9) com a profundidade comparativa da investigação de LIN et al. (12). É precisamente nesta lacuna que o método ColATeC, proposto neste trabalho, se insere, ao oferecer uma metodologia para a coleta e análise sistemática e comparativa do tempo de resposta a vulnerabilidades em um espectro abrangente de distribuições Linux, podendo ser estendida a outras coleções de *software*.

O *framework* DIFFCVSS, proposto por WU et al. (13), representa um avanço significativo na priorização de vulnerabilidades, permitindo uma reavaliação da severidade que leva em conta as especificidades de sistemas operacionais derivados. No entanto, o foco de seu método está na análise estática para a recalibragem do risco. O trabalho não avança para medir como essa severidade diferencial se traduz em ações práticas por parte dos mantenedores, como a velocidade na disponibilização de *patches*. Abre-se, assim, uma oportunidade de pesquisa complementar: uma vez que o DIFFCVSS ajuda a entender "o quão ruim" uma vulnerabilidade é para um sistema específico, torna-se essencial medir "o quão rápido" aquele sistema reage. O método ColATeC proposto aqui visa preencher exatamente essa lacuna, fornecendo a métrica temporal que complementa a análise de severidade contextual de WU et al. (13). Na implementação do banco de dados, os coletores preservam simultaneamente as classificações de severidade divulgadas por cada distribuição (por exemplo, "*Critical*" no Red Hat e "*Moderate*" no AlmaLinux) e os tempos de correção associados, permitindo verificar empiricamente se a diferenciação sugerida pelo DIFFCVSS se converte em priorizações distintas.

A compreensão da janela de vulnerabilidade evoluiu na literatura através de diferentes lentes investigativas. O conceito foi primeiramente formalizado por ARBAUGH, W. et al. (5), que propuseram um modelo de ciclo de vida para descrever os estados de uma vulnerabilidade. Anos depois, FREI et al. (14) levaram este conceito do plano teórico para o empírico, realizando uma análise em larga escala para quantificar a duração média entre a disponibilização de *patches* e o surgimento de *exploits*. Mais recentemente, BERES et al. (18) exploraram uma terceira via, a da simulação, modelando como diferentes políticas de segurança poderiam, em teoria, otimizar o fechamento dessa janela em um ambiente corporativo. O presente trabalho avança nesta linha evolutiva ao dissecar um dos eventos mais críticos do modelo de ARBAUGH, W. et al. (5), a disponibilização do *patch*. Em vez de tratá-lo como um evento monolítico, como em FREI et al. (14), ou um parâmetro em uma simulação, como em BERES et al. (18), o método ColATeC propõe medi-lo de forma

empírica e comparativa, revelando como essa disponibilização varia significativamente entre os diferentes ecossistemas das distribuições Linux. Dois eixos críticos emergem dessa análise: (i) estudos anteriores focaram em ecossistemas isolados (ex.: PyPI), enquanto este trabalho abrange múltiplas distribuições Linux; (ii) a dependência de procedimentos manuais limita a replicabilidade em grandes volumes de dados, a análise longitudinal contínua e a detecção de padrões ocultos.

2.2 Comparação com este trabalho

A literatura existente estabelece de forma robusta a importância crítica do tempo de correção de vulnerabilidades como uma métrica de segurança. Trabalhos anteriores realizaram análises valiosas em ecossistemas específicos, como o Debian, o Fedora e o PyPI, fornecendo informações importantes sobre seus processos de segurança.

Contudo, identificam-se as seguintes lacunas:

i) **Falta de um Método Padronizado**: Não existe na literatura um método sistemático e replicável para coletar, tratar e comparar continuamente dados de correção de vulnerabilidades entre múltiplas e distintas distribuições Linux. Os estudos existentes são, em geral, "fotografias" de um momento, com metodologias de coleta de dados específicas para os alvos analisados;

ii) **Escopo Limitado de Comparação**: Poucos trabalhos se dedicaram a comparar diretamente distribuições com filosofias e modelos de desenvolvimento distintos, como o Debian (baseado em comunidade) e as novas distribuições derivadas do RHEL (com forte influência corporativa);

iii) **Foco no "O Quê" e não no "Como"**: As pesquisas descrevem quais foram os tempos de correção, mas não propõem uma ferramenta ou um método que possa ser utilizado de forma contínua por gestores de TI e pesquisadores para extrair esses dados de forma autônoma.

Este trabalho se posiciona para preencher essas lacunas por meio do desenvolvimento do método ColATeC. Diferente das análises pontuais, o ColATeC é projetado como uma abordagem sistemática e extensível, cujo objetivo não é apenas apresentar um resultado, mas fornecer um processo para que essa análise possa ser replicada e expandida, fomentando a transparência e a melhoria contínua nos processos de segurança das distribuições de *software* livre, através de um processo automatizado com três contribuições fundamentais:

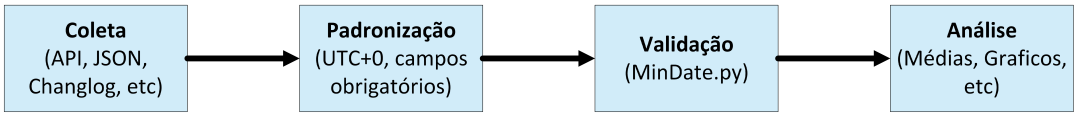
i) **Normalização de fluxos de dados**: Estabelecimento de padrões interoperáveis para publicação de informações sobre vulnerabilidades;

ii) **Automatização escalável**: Mecanismos adaptáveis para análise contínua em diversas distribuições Linux e outros *softwares*;

iii) **Transparência auditável**: Rastreabilidade técnica e temporal das métricas, assegurando verificação independente.

A figura 1 ilustra o fluxo básico do método ColATeC e logo após o detalhamento de cada etapa:

Figura 1 – Diagrama simplificado do fluxo de processamento do método ColATeC



Fonte: Produzido pelo autor

i) **Coleta:** É o *download* de dados estruturados através de *Application Programming Interface (API)*s, processamento de arquivos JSON, entre outros, contendo as informações necessárias para o ColATeC, que estão detalhadas na subseção 2.2.1;

ii) **Padronização:** É o tratamento/processamento dos dados, separando as informações e gravando em um banco de dados para processamento;

iii) **Validação:** Consiste na verificação automática da integridade dos dados coletados, checando a conformidade com o formato JSON esperado, a presença de todos os campos obrigatórios, além do uso do código MinDate.py, cuja função está detalhada na subseção 3.2.1.1;

iv) **Análise:** É a geração de gráficos, relatórios, e demais informações que possam ser úteis.

A subseção 3.2.1 apresenta detalhadamente o fluxo e o funcionamento do método ColATeC.

2.2.1 Organização do método ColATeC

A estrutura do ColATeC está organizada conforme abaixo:

i) **Dados estruturados:** Utiliza-se, preferencialmente, o formato JSON para padronização dos dados, com campos obrigatórios (figura 2);

ii) **Orientação para acesso aos dados:** No contexto deste trabalho, identificou-se a dificuldade de localização de dados estruturados, o que frequentemente exigiu a

navegação por múltiplas *Uniform Resource Locator (URL)s* ou acesso direto a arquivos do sistema operacional. Portanto, recomenda-se que as distribuições forneçam instruções claras para obtenção dos dados em, preferencialmente, um único arquivo estruturado. Caso necessário, é aceitável a requisição de registro para geração de chave de acesso. Um exemplo notável é a forma como Red Hat disponibiliza suas informações, fornecendo inclusive um código Python pronto para uso (19);

iii) **Padronização temporal:** Adoção obrigatória do fuso horário *Coordinated Universal Time (UTC)* (Greenwich) em todos os registros, evitando discrepâncias na interpretação de datas;

iv) **Definição de campos obrigatórios:** Os dados estruturados devem incluir:

- **Registro CVE**: Identificador único para comparação transversal da vulnerabilidade em diferentes ambientes;
- **Distribuição:** Nome oficial da distribuição Linux ou *software*;
- **Versão da Distribuição:** Versões afetadas pela vulnerabilidade;
- **Fuso Horário:** *UTC*+0 (Greenwich);
- **Severidade:** Classificação de risco conforme critérios técnicos da distribuição;
- **Data de Publicação:** Data oficial de divulgação pública do registro CVE;
- **Data de Resolução:** Data efetiva de correção;
- **Data de Descoberta:** Data mais antiga em que desenvolvedores tomaram conhecimento da existência da vulnerabilidade.

Figura 2 – Exemplo de arquivo JSON contendo dados do método ColATeC

```
{
    ....
    "FUSO": "UTC+0",
    "IDCVE": "CVE-123456",
    "DISTRIBUIÇÃO": "EXEMPLO",
    "VERSAO": "1",
    "SEVERIDADE": "IMPORTANTE",
    "PUBLICACAO": "01/01/2024",
    "DESCOBERTA": "30/09/2023",
    "RESOLUCAO": "01/01/2024"
    ....
}
```

Fonte: Produzido pelo autor

Novamente Red Hat se apresenta como um modelo a ser seguido, uma vez que seu arquivo no formato JSON contém, com exceção da data de descoberta, todas as informações necessárias para o método ColATeC em um único arquivo, conforme ilustrado na figura 3:

Figura 3 – Trecho de arquivo JSON fornecido por Red Hat


```
{
  "threat_severity" : "Moderate",
  "public_date" : "2023-01-17T00:00:00Z",
  "bugzilla" : {
    "description" : "mysql: Server: Optimizer unspecified vulnerability (CPU Jan 2023)",
    "id" : "2162273",
    "url" : "https://bugzilla.redhat.com/show_bug.cgi?id=2162273"
  },
  "cvss3" : {
    "cvss3_base_score" : "4.9",
    "cvss3_scoring_vector" : "CVSS:3.1/AV:N/AC:L/PR:H/UI:N/S:U/C:N/I:N/A:H",
    "status" : "verified"
  },
  "details" : [ "Vulnerability in the MySQL Server product of Oracle MySQL (component: Server
high privileged attacker with network access via multiple protocols to compromise MySQL Serve
repeatable crash (complete DOS) of MySQL Server. CVSS 3.1 Base Score 4.9 (Availability impact
  "affected_release" : [ {
    "product_name" : "Red Hat Enterprise Linux 8",
    "release_date" : "2022-10-25T00:00:00Z",
    "advisory" : "RHSA-2022:7119",
    "cpe" : "cpe:/a:redhat:enterprise_linux:8",
    "package" : "mysql:8.0-8060020220830124159.ad008a3a"
  }, {
    "product_name" : "Red Hat Enterprise Linux 9",
    "release_date" : "2022-09-20T00:00:00Z",
    "advisory" : "RHSA-2022:6590",
    "cpe" : "cpe:/a:redhat:enterprise_linux:9",
    "package" : "mysql-0:8.0.30-3.el9_0"
  }, {
    "product_name" : "Red Hat Software Collections for Red Hat Enterprise Linux 7",
    "release_date" : "2022-09-14T00:00:00Z",
    "advisory" : "RHSA-2022:6518",
    "cpe" : "cpe:/a:redhat:rhel_software_collections:3::el7",
    "package" : "rh-mysql80-mysql-0:8.0.30-1.el7"
  } ],
  "package_state" : [ {
```


Fonte: <https://access.redhat.com/hydra/rest/securitydata/cve/CVE-2023-21866.json> - Acessado em 10/08/2025

Analisando o fluxo do ColATeC representado pela figura 1, sua implementação pode ser realizada com qualquer linguagem de programação. Neste trabalho, foi utilizada a linguagem de programação Python devido à sua facilidade de uso e em cada etapa será indicado o código utilizado e sua função. A primeira questão desse trabalho (Q1: "*Quais informações são essenciais para a aplicação do método ColATeC?*") é respondida ao longo deste capítulo, que detalha os requisitos técnicos e metodológicos essenciais para sua implementação.

Capítulo 3

Metodologia

Este capítulo detalha a metodologia empregada para desenvolver e avaliar o método ColATeC, proposto para a análise de tempo para correções de vulnerabilidades em distribuições Linux. A metodologia foi estruturada para responder às questões de pesquisa apresentadas no capítulo 1, seguindo um delineamento de pesquisa empírica e quantitativa. São apresentados o método proposto, a definição formal das métricas utilizadas, o desenho experimental e uma discussão sobre as ameaças à validade deste estudo.

3.1 Uso de inteligência artificial (IA)

Durante a elaboração deste trabalho, recorreu-se ao uso de ferramentas de inteligência artificial generativa como suporte em atividades específicas. Foram utilizadas as plataformas DeepSeek (versões V3 e R1), Google Gemini 2.5 Pro e OpenAI GPT-5-Codex, principalmente para: (1) revisão textual e adequação às normas da Associação Brasileira de Normas Técnicas (ABNT); e (2) análise de código-fonte, desenvolvimento de *scripts* para processamento de dados e interpretação preliminar de resultados durante a fase de estruturação metodológica.

Ressalta-se que todos os conteúdos gerados por IA passaram por rigorosa validação, incluindo verificação de precisão factual e consistência com a literatura da área. As etapas de interpretação, análise, síntese e as contribuições intelectuais deste trabalho permanecem de responsabilidade exclusiva do autor, que realizou revisão e validação integral do material produzido.

3.2 O método ColATeC

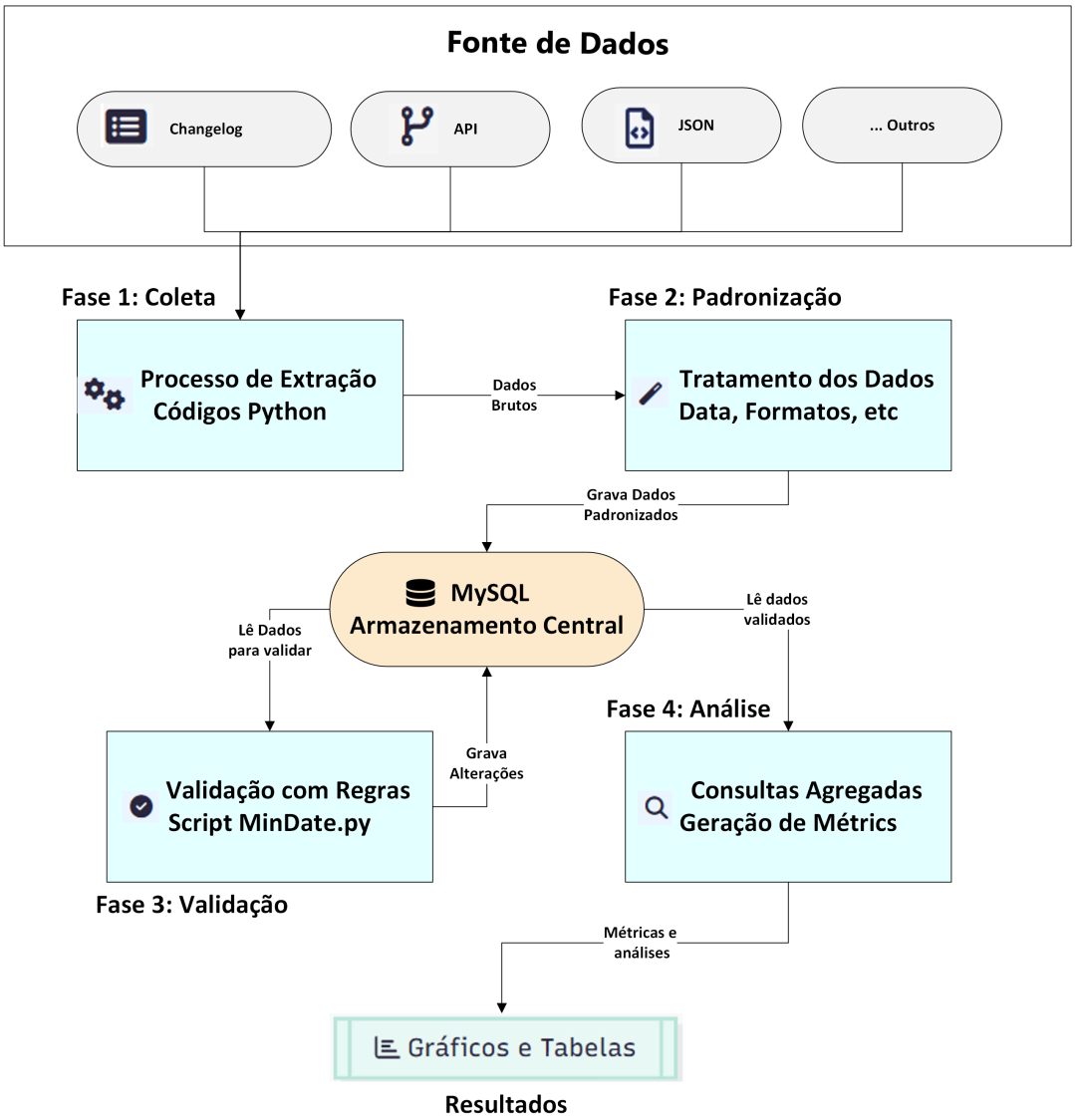
Nesta seção, descreve-se a arquitetura e o funcionamento do método ColATeC, desenvolvido como a principal ferramenta para a execução deste trabalho.

3.2.1 Arquitetura e fluxo de execução

A arquitetura do ColATeC foi concebida de forma modular para garantir extensibilidade e manutenibilidade do sistema. Conforme ilustrado na figura 1, o fluxo de execução compreende quatro etapas principais e sequenciais: coleta, padronização, validação e análise.

A figura 4 apresenta o fluxograma das quatro fases do ColATeC e suas interconexões. Posteriormente, detalha-se cada uma dessas etapas.

Figura 4 – Diagrama simplificado do fluxo completo do método ColATeC



Fonte: Produzido pelo autor

A primeira etapa, de coleta de dados, é implementada por meio de um conjunto de módulos de *software* desenvolvidos em Python. Uma das principais dificuldades abordadas neste trabalho é a ausência de um padrão unificado para a disponibilização de dados de segurança entre as diferentes distribuições Linux. Esta heterogeneidade exigiu o desenvolvimento de coletores personalizados para cada família de distribuição, conforme detalhado no capítulo 4.

A segunda etapa, a padronização, é um processo crucial que transforma os dados brutos e heterogêneos, obtidos na etapa de coleta, em um formato coeso, estruturado e consistente. Esta fase é essencial para viabilizar a análise comparativa entre as distribuições e é executada através de três atividades principais:

– **Extração e Normalização de Dados:** Cada dado coletado, seja de uma API em formato JSON ou de uma página web via *web scraping*, passa por um processo de análise para extrair as informações essenciais. Durante esta atividade, os dados são normalizados para um padrão comum. Um exemplo notável é a normalização de datas, que podem vir em diversos formatos (e.g., “Aug 12, 2025”, “2025/08/12”). Todas as datas são convertidas para o padrão **ANO-MÊS-DIA** (YYYY-MM-DD) para garantir a consistência nos cálculos temporais. Da mesma forma, nomes de pacotes e versões são limpos e padronizados. Um exemplo desses registros no banco de dados é apresentado pelas figuras 5 e 6;

Figura 5 – Exemplo dos registros no banco de dados

CVE	Version	Published	Published_NIST	Resolved	FixState
CVE-2023-6879	Red Hat Enterprise Linux 8	2023-12-28	2023-12-27		not affected
CVE-2023-6879	Red Hat Enterprise Linux 8	2023-12-28	2023-12-27		not affected
CVE-2023-6879	Red Hat Enterprise Linux 8	2023-12-28	2023-12-27		not affected
CVE-2023-6879	Red Hat Enterprise Linux 9	2023-12-28	2023-12-27		not affected
CVE-2023-6879	Red Hat Enterprise Linux 9	2023-12-28	2023-12-27		not affected
CVE-2023-6879	Red Hat Enterprise Linux 9	2023-12-28	2023-12-27		not affected
CVE-2023-6879	Red Hat Enterprise Linux 9	2023-12-28	2023-12-27		not affected
CVE-2023-51767	Red Hat Enterprise Linux 6	2023-12-24	2023-12-24		out of supp
CVE-2023-51767	Red Hat Enterprise Linux 7	2023-12-24	2023-12-24		out of supp
CVE-2023-51767	Red Hat Enterprise Linux 8	2023-12-24	2023-12-24		will not fix
CVE-2023-51767	Red Hat Enterprise Linux 9	2023-12-24	2023-12-24		will not fix
CVE-2023-51714	Red Hat Enterprise Linux 6	2023-12-24	2023-12-24		out of supp
CVE-2023-51714	Red Hat Enterprise Linux 6	2023-12-24	2023-12-24		out of supp
CVE-2023-51714	Red Hat Enterprise Linux 7	2023-12-24	2023-12-24		out of supp
CVE-2023-51714	Red Hat Enterprise Linux 7	2023-12-24	2023-12-24		out of supp
CVE-2023-51714	Red Hat Enterprise Linux 8	2023-12-24	2023-12-24	2024-05-22	
CVE-2023-51714	Red Hat Enterprise Linux 9	2023-12-24	2023-12-24	2024-04-30	

Fonte: Produzido pelo autor

Figura 6 – Exemplo dos registros no banco de dados - padronização dos nomes de pacotes

FixState	Severity	Severity_NIST	Package	NormPackage
not affected	important	CRITICAL	firefox	firefox
not affected	important	CRITICAL	firefox:flatpak/firefox	firefox
not affected	important	CRITICAL	thunderbird	thunderbird
not affected	important	CRITICAL	thunderbird:flatpak/thunderbird	thunderbird
out of support scope	moderate	HIGH	openssh	openssh
out of support scope	moderate	HIGH	openssh	openssh
will not fix	moderate	HIGH	openssh	openssh
will not fix	moderate	HIGH	openssh	openssh
out of support scope	moderate	CRITICAL	qt	qt
out of support scope	moderate	CRITICAL	qt3	qt3
out of support scope	moderate	CRITICAL	qt	qt
out of support scope	moderate	CRITICAL	qt3	qt3
	moderate	CRITICAL	qt5-qtbase-0:5.15.3-7.el8	qt5-qtbase
	moderate	CRITICAL	qt5-qtbase-0:5.15.9-9.el9	qt5-qtbase
not affected	important	HIGH	kernel	kernel

Fonte: Produzido pelo autor

- **Definição de um Esquema de Dados Unificado:** Para armazenar as informações de forma estruturada, foi projetado um esquema de banco de dados relacional. Os registros foram armazenados em um banco de dados MySQL, denominado **cvedb5**, cujo esquema relacional está detalhado no arquivo Database.txt. Este esquema define as entidades centrais do problema, como vulnerabilidades (armazenando dados da CVE), distribuições (catalogando os sistemas analisados), avisos (com os metadados dos *advisories*) e pacotes, e os relacionamentos entre elas. Este modelo de dados unificado é a espinha dorsal do ColATeC, permitindo correlacionar uma única CVE a múltiplos avisos e pacotes em diferentes distribuições;

- **Mapeamento e Carga:** Após a normalização, os dados são mapeados para o esquema unificado e carregados no banco de dados. Este processo, análogo a um fluxo de *Extract, Transform, Load (ETL)*, garante que, independentemente da origem ou do formato inicial, toda informação relevante ocupe seu devido lugar na estrutura de dados final. Por exemplo, a data de um *Red Hat Security Advisory (RHSA)* e de um *Debian Security Advisor (DSA)* são ambas mapeadas para o mesmo campo no banco de dados, permitindo comparações diretas.

Ao final desta etapa, obtém-se um repositório de dados unificado, normalizado e consistente, que servirá como base confiável para as fases subsequentes de validação e análise.

Tabela 2 – Resumo entidade-relacionamento do banco **cvedb5**

Entidade	Descrição	Relacionamentos Principais
CVE	Armazena metadados canonizados da CVE (identificador, datas do NVD, severidade base, referência oficial).	Relaciona-se com ***advisory*** (1:n) por meio da chave CVE; fornece o ponto de união entre distribuições distintas.
distribution	Catálogo de distribuições tratadas pelo método, incluindo nome, ecossistema (RPM/DEB) e indicadores de suporte.	Referenciada por ***advisory*** (n:1) e ***package*** (n:1) para permitir filtragem por família.
advisory	Representa cada aviso de segurança coletado (RHSA, DSA, USN etc.), com datas de publicação e resolução padronizadas.	Liga-se a ***package*** (1:n) para detalhar os pacotes corrigidos e herda a severidade específica da distribuição.
package	Normaliza nomes de pacotes por meio de identificadores canônicos, mantendo versões afetadas e corrigidas.	Conecta-se a ***advisory*** (n:1) e, indiretamente, à **CVE** para suportar análises por software comum entre ecossistemas.
cvemindate	Guarda a menor data pública identificada para cada CVE, calculada pelo *script* **MinDate.py**.	Relacionamento (1:1) com **CVE**; utilizado para recalibrar o marco temporal T_0 nas estatísticas.

Fonte: Produzido pelo autor

A terceira etapa, a validação, tem como objetivo garantir a qualidade, a integridade e a consistência lógica dos dados antes de proceder à análise. Esta fase atua como um controle de qualidade automatizado, prevenindo que erros de coleta ou padronização resultem em conclusões equivocadas. As rotinas de validação incluem:

- **Verificação de Completude:** O sistema verifica se os registros essenciais para o cálculo da métrica principal estão presentes. Por exemplo, um evento de correção só é considerado válido se possuir tanto o marco inicial (T_0, data da CVE) quanto o marco final (T_1, data da correção). Eventos incompletos são tratados conforme subseção 3.4.2.1 e subseção 3.4.2.3;

- **Validação de Consistência Temporal:** É executada uma verificação lógica para assegurar que a data do aviso de correção (T_1) não seja anterior à data de publicação da CVE (T_0). Casos que violam esta regra podem ocorrer devido a erros de digitação nas fontes de dados ou a processos de divulgação embargados e são investigados manualmente ou descartados para não distorcer os resultados, conforme subseção 3.4.2.2;

– **Detecção de Duplicatas:** O código utilizado neste trabalho implementa mecanismos para identificar e tratar registros duplicados que possam ter sido inseridos durante a etapa de coleta, garantindo que cada evento de correção seja contabilizado apenas uma vez. É importante ressaltar que uma mesma CVE pode afetar diversas vezes a mesma distribuição Linux, seja em versões diferentes ou uma CVE que atinja vários pacotes ao mesmo tempo e, neste caso, não se trata de uma duplicata e o registro foi inserido na base de dados.

3.2.1.1 O Desafio da Determinação do Marco Inicial: Um Estudo de Caso

No caso da validação de consistência temporal, a figura 1 faz referência ao código MinDate.py. Este código foi desenvolvido pois, após uma análise detalhada dos dados coletados pelo ColATeC, revelou-se um fenômeno recorrente e de grande importância: a existência de correções de vulnerabilidades aplicadas em data anterior à divulgação oficial da CVE. Este padrão pode ser ilustrado pelo caso da **CVE-2023-21866**, uma vulnerabilidade no MySQL *Server* da Oracle. Conforme o registro do NIST, a data de publicação desta CVE foi 17 de Janeiro de 2023[1]. No entanto, como evidencia a tabela 3, as distribuições Red Hat e Ubuntu já haviam disponibilizado correções para esta falha antes dessa data.

Tabela 3 – Dados comparativos da CVE-2023-21866

Distribuição	Versão	Publicação	Correção	Referência
Red Hat	8	16/01/2023	25/10/2022	CVE-2023-21866[2]
Red Hat	9	16/01/2023	20/09/2022	CVE-2023-21866[2]
Ubuntu[3]	20.04	18/01/2023	02/05/2022	CVE-2023-21866[4]
Ubuntu[3]	22.04	18/01/2023	26/04/2022	CVE-2023-21866[5]
Oracle	—	17/01/2023	17/01/2023	CVE-2023-21866[6]

Fonte: Produzido pelo autor

Este cenário é característico do processo de *Coordinated Vulnerability Disclosure (CVD)* formalizado nos padrões *International Organization for Standardization (ISO)* e pela *International Electrotechnical Commission (IEC)*, através da ISO/IEC 29147:2018 (20). A CVD representa um modelo de segurança colaborativo que alavanca uma rede global de pesquisadores de segurança independentes para identificar falhas antes que sejam exploradas por atores maliciosos, conforme WALSHE e SIMPSON (21). O processo estabelece

[1] <https://nvd.nist.gov/vuln/detail/cve-2023-21866> (Acessado em 19 de Agosto de 2025).
[2] <https://access.redhat.com/security/cve/CVE-2023-21866> (Acessado em 01/09/2024)
[3] Informações das versões corrigidas obtidas em <https://ubuntu.com/security/CVE-2023-21866> (Acessado em 01/09/2024)
[4] <https://launchpad.net/ubuntu/+source/mysql-8.0/8.0.29-0ubuntu0.20.04.2> (Acessado em 01/09/2024)
[5] <https://launchpad.net/ubuntu/+source/mysql-8.0/8.0.29-0ubuntu0.22.04.1> (Acessado em 01/09/2024)
[6] <https://www.oracle.com/security-alerts/cpujan2023.html> (Acessado em 01/09/2024)

um fluxo de comunicação privada entre o pesquisador e o fornecedor, garantindo um período de embargo para o desenvolvimento da correção. A divulgação pública da CVE é então coordenada para coincidir com a disponibilização das correções, visando minimizar a janela de exposição para os usuários. Apesar de a CVD ser uma prática estabelecida e benéfica, ela expõe a limitação fundamental da métrica tempo de correção utilizada na literatura: a dependência da data de publicação como marco inicial (T_0), em detrimento da verdadeira data de descoberta pelo fornecedor.

A eficácia de um programa de CVD de um fornecedor, conforme discutido por WALSHE e SIMPSON (21), depende de diversos fatores, incluindo a agilidade de sua equipe interna para processar, verificar e corrigir as falhas reportadas. Nesse contexto, a métrica tempo de correção, calculada pelo ColATeC, pode ser interpretada como um indicador quantitativo da eficiência da fase de resposta interna de um fornecedor ao seu programa de CVD. Longos tempos de correção podem sugerir gargalos ou ineficiências nesse processo.

Diante deste cenário, este trabalho argumenta que a transparência e a precisão da análise de segurança seriam imensamente enriquecidas pela adoção de uma nova política de divulgação. O ColATeC, ao evidenciar estas discrepâncias temporais, fundamenta a proposta de que as instituições (públicas ou privadas) e os pesquisadores envolvidos no ciclo de vida das vulnerabilidades passem a registrar e divulgar a data de descoberta sempre que esta for conhecida por meios legítimos (e.g., descoberta interna, programas de recompensa por *bugs*). A existência deste dado, mesmo que a correção ocorra meses ou anos depois, permitiria a criação de métricas de segurança mais realistas e transparentes, distinguindo o tempo de reconhecimento da falha do tempo de mobilização para a correção.

Para mitigar essa lacuna neste trabalho e evitar inconsistências temporais (como datas negativas), padronizou-se o cálculo das médias de exposição ao risco, garantindo a consistência metodológica. O código MinDate.py foi desenvolvido para identificar a menor data de publicação disponível para cada vulnerabilidade. Conforme evidenciado pela tabela 3, uma mesma vulnerabilidade (CVE) pode apresentar datas de publicação distintas em diferentes fontes. Diante disso, adotou-se como válida a menor data identificada, aplicando-a de forma uniforme a todas as distribuições para aquela CVE específica. Dessa forma, assegura-se que a análise seja realizada com base na data mais antiga conhecida. Após a extração e processamento, os resultados são armazenados na tabela **cvemindate** do banco de dados.

A quarta e última etapa, a análise de dados, é onde os dados validados são efetivamente utilizados para responder às questões de pesquisa. Esta fase é executada através de um conjunto de análises descritivas e comparativas, realizadas com o auxílio de consultas *Structured Query Language (SQL)* diretamente no banco de dados e com códigos-fonte em Python para a geração de estatísticas e visualizações. As principais análises realizadas incluem: o cálculo da métrica principal de tempo de correção para cada evento; a aplicação

de estatística descritiva (média, mediana, desvio padrão) para caracterizar o comportamento de cada distribuição; e, finalmente, a consolidação dos achados em visualizações gráficas para facilitar a interpretação e a comunicação dos resultados.

3.3 Definição das métricas

Para garantir a clareza e a replicabilidade do estudo, é fundamental definir de forma inequívoca as métricas utilizadas.

3.3.1 Tempo de correção de vulnerabilidade

A principal métrica deste trabalho é o tempo de correção de vulnerabilidade. Esta métrica busca quantificar o intervalo entre o momento em que uma vulnerabilidade se torna publicamente conhecida e o momento em que uma distribuição específica disponibiliza oficialmente uma correção para seus usuários.

Para seu cálculo, foram definidos os seguintes marcos:

- **Marco Inicial (T_0):** Corresponde à menor data descoberta pelo MinDate.py;
- **Marco Final (T_1):** Corresponde à data de emissão do *security advisory* (aviso de segurança) oficial pela equipe de segurança da distribuição analisada. Este marco foi escolhido por representar o momento em que o administrador de sistemas é formalmente notificado da disponibilidade de uma correção e pode tomar uma ação corretiva.

Dessa forma, o tempo de correção é calculado pela seguinte equação:

$$\text{Tempo de Correção} = T_1 - T_0$$

Reconhece-se que uma correção pode existir em repositórios de teste ou *branch* de desenvolvimento antes do marco T_1. No entanto, a métrica definida reflete a perspectiva do usuário final e do administrador de sistemas, para quem a correção só é considerada disponível após o anúncio oficial. Em casos onde T_0 for igual a T_1, será considerado o valor de 1 dia.

Os gráficos anuais apresentados no capítulo 5 utilizam o campo ***Year***, derivado diretamente do atributo *Published* fornecido pelo NVD. Essa escolha garante consistência com o ponto de partida formal das análises e evita ambiguidades com datas calculadas posteriormente pelo **MinDate.py**.

3.4 Delineamento experimental

Para avaliar o desempenho das distribuições, foi desenhado um experimento de larga escala, cujos detalhes são apresentados a seguir.

3.4.1 Seleção das distribuições analisadas

A seleção das distribuições Linux para este estudo não foi arbitrária e baseou-se em um conjunto de critérios que visam garantir a relevância e a representatividade dos resultados. Os critérios foram:

- **Relevância de Mercado e Popularidade:** Foram escolhidas distribuições com alta penetração em servidores e *desktops*, como as famílias Red Hat e Debian, que dominam o mercado corporativo e comunitário;
- **Representatividade de Ecossistemas:** As distribuições selecionadas representam os dois maiores e mais influentes ecossistemas do mundo Linux: o ecossistema RPM (representado por Red Hat, AlmaLinux e Rocky Linux) e o ecossistema DEB (representado por Debian e Ubuntu);
- **Diversidade de Modelos de Desenvolvimento:** A amostra inclui distribuições com diferentes modelos de governança e lançamento, como o modelo estritamente comercial da Red Hat, o modelo comunitário do Debian, AlmaLinux e Rocky Linux, e o modelo híbrido do Ubuntu;
- **Relevância de Derivados (Caso de Estudo):** A inclusão de AlmaLinux e Rocky Linux justifica-se pelos seus papeis como sucessores diretos do CentOS, tornando a análise de seu comportamento de segurança um caso de estudo de grande interesse para a comunidade.

3.4.2 Fontes de dados

As fontes de dados primárias para o método ColATeC são essenciais para a determinação dos marcos temporais do estudo. As fontes utilizadas são:

- **NVD:** Utilizado para obter a lista de CVEs a serem analisadas e suas respectivas datas de publicação. Esta data é uma das fontes candidatas para a determinação do marco inicial do evento de vulnerabilidade (T_0);
- **Avisos de Segurança Oficiais:** Coletados diretamente dos portais de segurança de cada distribuição (e.g., Red Hat *Security Advisories*, Debian *Security Advisories*). As datas de publicação destes avisos também são consideradas como candidatas para o marco T_0, enquanto a data em que a vulnerabilidade é efetivamente corrigida, conforme informado no aviso, é utilizada para definir o marco final (T_1).

Para garantir a medição mais precisa do tempo de exposição, o marco T_0 é formalmente definido como a data mais antiga entre todas as datas de publicação disponíveis para uma mesma CVE, seja ela proveniente do NVD ou dos diversos avisos de segurança. Este processo garante que a contagem do tempo inicie a partir do primeiro momento em que a vulnerabilidade se tornou pública em qualquer uma das fontes estudadas.

3.4.2.1 Tratamento de vulnerabilidades não corrigidas

Um desafio metodológico importante é o tratamento de vulnerabilidades que permanecem sem correção ao final do período de análise (31 de Dezembro de 2023). Atribuir uma data de correção artificial para esses casos introduziria um viés significativo, subestimando o tempo médio de resposta. Portanto, para garantir a validade dos resultados, esses eventos são considerados não corrigidos. Na prática, isso significa que, para o cálculo de estatísticas como média e mediana do tempo de correção, são considerados apenas os eventos que possuem um marco final (T_1) bem definido. As vulnerabilidades não corrigidas são analisadas separadamente na seção 5.4, mas não são incluídas no cálculo da velocidade de correção para não distorcer a análise.

3.4.2.2 Tratamento de vulnerabilidades com datas negativas

As vulnerabilidades que apresentaram datas negativas correspondem a *outliers*, ou seja, valores observados em um conjunto de dados que se desviam significativamente da norma ou padrão da maioria dos dados. Estes dados não serão incluídos nas estatísticas das distribuições, pois comprometem os resultados. Por este motivo, também serão analisados na seção 5.4.

3.4.2.3 Tratamento de vulnerabilidades com erro de coletada

As vulnerabilidades que apresentaram erros durante a coleta, seja pela dificuldade na obtenção dos dados ou por problemas técnicos, não serão incluídas nas estatísticas das distribuições, pois comprometem os resultados. Por este motivo, será analisada a porcentagem de vulnerabilidades descartadas e apresentadas à parte em cada distribuição.

3.4.3 Período de coleta e análise

O estudo analisará as vulnerabilidades (CVE) que afetam as distribuições selecionadas e que foram publicadas no NVD durante o período de 1 de Janeiro de 2019 a 31 de Dezembro de 2023. Este período foi escolhido para garantir um volume de dados significativo e a relevância temporal dos resultados. Estudos futuros que utilizarem o ColATeC podem considerar outros períodos.

3.5 Ameaças à validade

Todo estudo empírico possui limitações. Nesta seção, discutem-se as potenciais ameaças à validade deste trabalho e as estratégias adotadas para mitigá-las, em um exercício de rigor científico e transparência.

3.5.1 Ameaças à validade de construção

Refere-se ao grau em que a métrica tempo de correção realmente representa a postura de segurança de uma distribuição. A principal ameaça é que a data do aviso oficial pode não refletir o momento exato em que uma correção se tornou disponível em um repositório. **Mitigação:** Reconhece-se esta limitação, porém a métrica é válida sob a perspectiva do usuário. A consistência na aplicação desta métrica para todas as distribuições garante uma base de comparação justa.

3.5.2 Ameaças à validade interna

Refere-se a fatores não controlados que podem influenciar os resultados, como falhas no *software* de coleta ou disponibilização dos dados. **Mitigação:** O código-fonte do ColATeC foi submetido a revisões e testes manuais sob amostras de algumas dezenas de registros. Além disso, os dados coletados passam por um processo de limpeza e verificação de consistência para identificar anomalias (e.g., T_1 anterior a T_0). Os erros nas coletas foram tratados conforme subseção 3.4.2.3.

3.5.3 Ameaças à validade externa

Refere-se à capacidade de generalizar os resultados. Os achados deste estudo são diretamente aplicáveis às versões e distribuições analisadas dentro do período de tempo definido. **Mitigação:** Ao escolher distribuições que representam os maiores ecossistemas, aumenta-se a probabilidade de que os padrões observados sejam indicativos de comportamentos mais amplos. No entanto, declaramos explicitamente que a generalização para outras distribuições (como Arch ou SUSE) deve ser feita com cautela.

3.5.4 Limitações das fontes de dados

O estudo depende de fontes de dados públicas e oficiais. Vulnerabilidades discutidas apenas em listas de e-mail privadas, *bug trackers* restritos ou que não recebem um CVE não são cobertas. **Mitigação:** Esta é uma limitação inerente a estudos de larga escala. O uso das fontes mais canônicas e globais (NVD) e dos canais oficiais de cada distribuição garante que estamos analisando o conjunto de vulnerabilidades mais relevante e universalmente reconhecido.

Capítulo 4

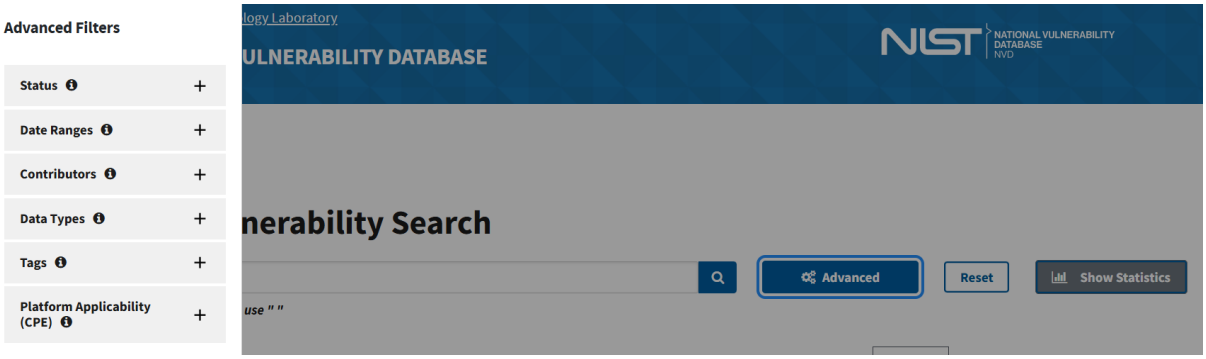
Estudo de caso - coleta dos dados

Este capítulo apresenta a aplicação do método ColATeC nas distribuições Linux estudadas. São avaliados os principais canais de comunicação das vulnerabilidades, o processo de obtenção e o volume de dados coletados de cada fonte. Adicionalmente, detalham-se o processo de coleta, os desafios técnicos superados e as características do conjunto de dados abrangente utilizado neste trabalho.

4.1 NIST

O NIST divulga as vulnerabilidades por meio de sua base de dados NVD (22). Essa plataforma permite consultas personalizadas com filtros avançados, sendo acessível tanto para especialistas em segurança quanto para usuários com menor familiaridade no tema. A figura 7 ilustra a interface de pesquisa avançada da NVD:

Figura 7 – Opções de filtros de uma busca avançada do NIST



Fonte: NVD (23)

A extração de dados foi automatizada utilizando a biblioteca **nvdlib** (24) para Python, que atua com a interface **API REST 2.0** do NIST (25), permitindo a automação na coleta de dados. Para utilizar esta API, é necessário solicitar uma chave de acesso à plataforma (26). O principal desafio técnico enfrentado nesta etapa foi a restrição de taxa de consulta (*rate limiting*) e a ocorrência de *timeouts* impostas pela API. Requisições que solicitavam um grande volume de dados de uma só vez (como todas as CVEs publicadas em um ano) excediam os limites de tempo do servidor, o que resultava em respostas incompletas e, consequentemente, na perda de dados.

Para garantir a integridade e a completude do conjunto de dados, foi necessário implementar uma estratégia de consulta iterativa e particionada. Em substituição a consultas únicas abrangentes, o coletor do ColATeC (find_CVE_NIST_5.py) foi adaptado para requisitar dados em intervalos mensais. Embora esta abordagem eleve o número total de requisições à API, ela revelou-se robusta e eficaz para evitar *timeouts*, assegurando que os dados extraídos do NVD constituíssem uma representação fidedigna e integral das vulnerabilidades publicadas.

Através deste processo metodológico, foram coletados e processados um total de **171.953** registros de CVEs do NVD, publicados durante o período estudado e armazenados na tabela **nist** do banco de dados. O desempenho médio, calculado com base na coleta de 100 registros, foi de 0,000938 segundo por registro.

4.2 Linux Red Hat

A distribuição Red Hat Linux foi analisada inicialmente, considerando que AlmaLinux (27) e Rocky Linux (28) são alternativas de código aberto compatíveis com seu ecossistema. Os comunicados de atualizações de segurança são denominados errata (29) e são categorizados da seguinte forma:

***Red Hat Security Advisory (RHSA)*:** Anúncios que descrevem correções críticas de segurança, podendo incluir ajustes secundários de *bugs* ou melhorias. Constituem a categoria de maior prioridade;

***Red Hat Bug Advisory (RHBA)*:** Anúncios focados em correções de erros funcionais, eventualmente acompanhadas de aprimoramentos. Classificados como segunda prioridade;

***Red Hat Enhancement Advisory (RHEA)*:** Anúncios dedicados exclusivamente a melhorias de desempenho ou funcionalidades.

Para a coleta de dados, utilizou-se a API oficial da Red Hat (19), cuja documentação oferece um exemplo de código em Python para consultas ao banco de vulnerabilidades

em formato JSON (30). Este modelo serviu de base para o código desenvolvido neste trabalho (arquivo api_redhat_5.py).

Não houve grandes desafios técnicos enfrentados nesta etapa, sendo que o Linux Red Hat foi a distribuição que apresenta a melhor experiência entre a documentação das informações e a coleta de dados.

Através deste processo metodológico, foram coletados e processados um total de **85.055** registros de CVEs Linux Red Hat, publicados durante o período estudado e armazenados na tabela **redhat** do banco de dados. Com exceção do campo data de descoberta (não fornecido pela Red Hat), todos os demais campos exigidos pelo método ColATeC estão presentes nos registros. O desempenho médio, calculado com base na coleta de 100 registros, foi de 0,284195 segundo por registro.

4.3 AlmaLinux

A distribuição AlmaLinux disponibiliza informações sobre segurança em seu portal oficial (31), incluindo a localização dos dados em formato JSON, opções para inscrição em listas de e-mail de alertas de segurança e outras ferramentas. Assim como Red Hat, AlmaLinux utiliza o termo errata (32) para referir-se aos comunicados de correções de vulnerabilidades e *bugs*, categorizados da seguinte forma:

***AlmaLinux Security Advisory (ALSA)*:** Anúncios que descrevem correções críticas de segurança, podendo incluir ajustes secundários de *bugs* ou melhorias;

***AlmaLinux Bug Advisory (ALBA)*:** Anúncios focados em correções de erros funcionais, eventualmente acompanhadas de aprimoramentos. Classificados como segunda prioridade;

***AlmaLinux Enhancement Advisory (ALEA)*:** Anúncios dedicados exclusivamente a melhorias de desempenho ou funcionalidades.

Para a coleta automatizada de dados, acessaram-se os arquivos JSON das versões 8 (33) e 9 (34) por meio do código desenvolvido para este trabalho (api_almalinux_5.py). Não houve grandes desafios técnicos enfrentados nesta etapa.

Através deste processo metodológico, foram coletados e processados um total de **6.580** registros de CVEs do AlmaLinux, publicados durante o período estudado e armazenados na tabela **almalinux** do banco de dados. Com exceção do campo data de descoberta (não fornecido pelo AlmaLinux), todos os demais campos exigidos pelo método ColATeC estão presentes nos registros. O desempenho médio, calculado com base na coleta de 100 registros, foi de 0,000605 segundo por registro.

4.4 Rocky Linux

A distribuição Rocky Linux disponibiliza informações sobre correções de vulnerabilidades e *bugs* em seu portal oficial (35), também adotando o termo errata para referir-se aos comunicados de correções de vulnerabilidades e *bugs*, categorizados da seguinte forma:

***Rocky Linux Security Advisories (RLSA)*:** Anúncios que descrevem correções críticas de segurança, podendo incluir ajustes secundários de *bugs* ou melhorias. Constituem a categoria de maior prioridade;

***Rocky Linux Bugfix Advisories (RLBA)*:** Anúncios focados em correções de erros funcionais, eventualmente acompanhadas de aprimoramentos. Classificados como segunda prioridade;

***Rocky Linux Enhancement Advisories (RLEA)*:** Anúncios dedicados exclusivamente a melhorias de desempenho ou funcionalidades.

A distribuição Rocky Linux não fornece os repositórios para busca dos arquivos via API ou no formato JSON. A ausência de documentação oficial exigiu a identificação manual de sua localização examinando as informações do link ***Errata API Endpoint OpenAPI Specs*** (35), e somente assim foi possível identificar a localização dos arquivos no formato JSON (36). O parâmetro **page=0** foi tratado dinamicamente no código desenvolvido para este trabalho (arquivo api_rockylinux_5.py), permitindo a coleta sequencial das páginas subsequentes (**page=1**, **page=2**, e assim por diante).

Um desafio observado foi a ausência da data de publicação nos registros JSON. Para contornar essa limitação, utilizou-se a data correspondente ao mesmo CVE na tabela **redhat** do banco de dados. Essa abordagem baseou-se na premissa de que o Rocky Linux, sendo uma distribuição derivada do Red Hat, possui os mesmos CVEs que o Linux Red Hat.

O principal desafio técnico enfrentado nesta etapa foi a falta de documentação clara de como acessar as informações, falta de dados importantes (sendo necessário coletar a data de publicação do Linux Red Hat) e a necessidade de navegar em diversas URLs gerou demora na coleta. Através deste processo, foram coletados e processados um total de **2.933** registros de CVEs do Linux Rocky Linux publicados durante o período estudado, que foram armazenados na tabela **rockylinux** do banco de dados. Com exceção do campo data de descoberta e data de publicação (não fornecido pelo Rocky Linux), todos os demais campos exigidos pelo método ColATeC estão presentes nos registros. O desempenho médio, calculado com base na coleta de 100 registros, foi de 0,000033 segundo por registro.

4.5 Debian

A distribuição Debian utiliza o rastreador de falhas de segurança security-tracker (37), que permite consultar vulnerabilidades em pacotes das versões estáveis e em testes. Por meio de sua interface, é possível verificar o estado das vulnerabilidades (resolvidas ou pendentes) em cada versão do sistema. Os dados brutos estão disponíveis em formato JSON (38), utilizados como fonte primária para este trabalho.

A estrutura do arquivo JSON do Debian organiza-se por pacotes, onde cada entrada agrega todas as CVEs relacionadas. No entanto, observou-se a ausência de campos críticos para o método ColATeC:

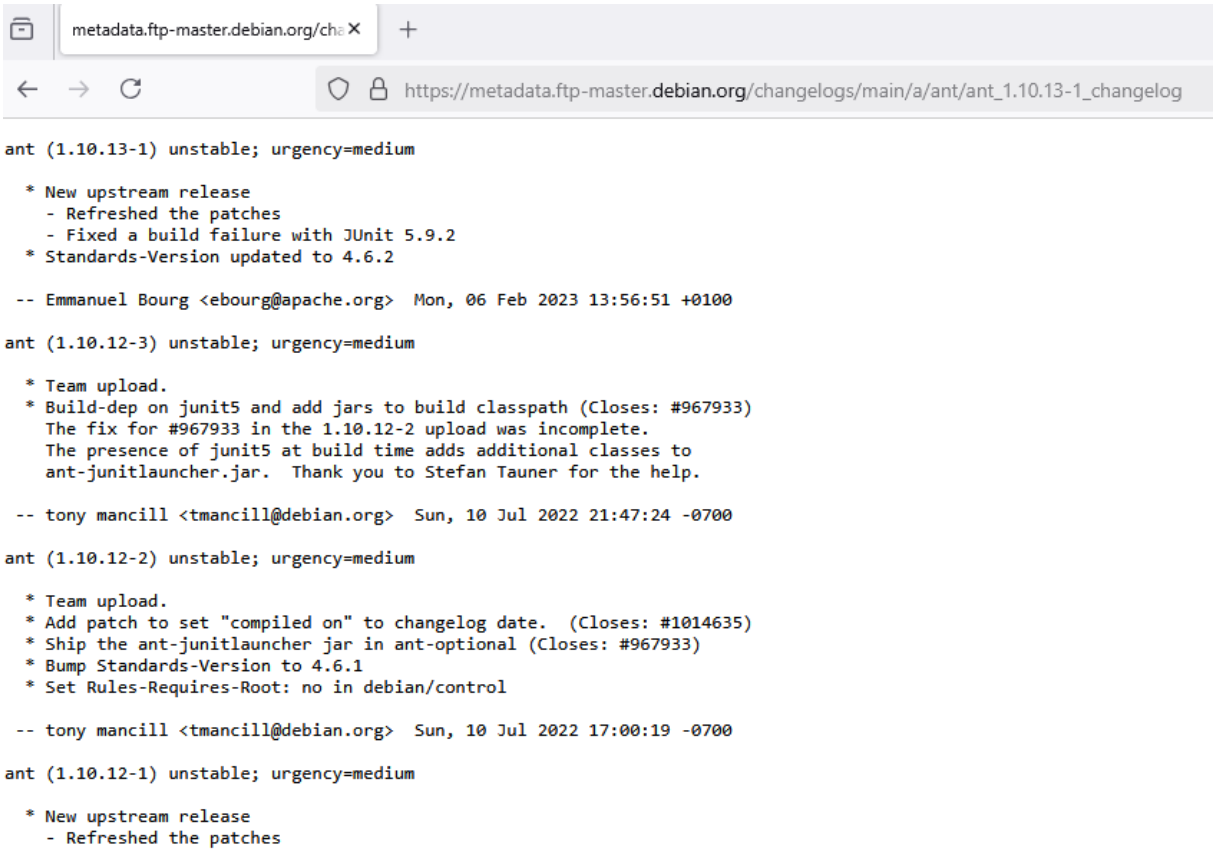
- **Data de publicação:** Não presente no JSON, exigindo a adoção da data mais antiga identificada durante a análise (conforme descrito na subseção 3.2.1.1);
- **Data de resolução:** Extraída de *changelogs* públicos, acessados programaticamente via URLs disponíveis nos registros;
- **Data de descoberta:** Não disponível na estrutura de dados.

Para obter o *changelog* remoto, identificou-se que o padrão da URL segue o formato: `https://metadata.ftp-master.debian.org/changelogs/main/letra/pacote/pacote_versao_changelog` onde:

- **`letra:`** Primeira letra do pacote Debian associado à CVE (exemplo: “a” para o pacote “apache2”);
- **`pacote:`** Nome do pacote Debian associado à CVE;
- **`versao:`** Versão na qual a vulnerabilidade foi corrigida.

A figura 8 ilustra um *changelog* acessado manualmente, mostrando detalhes de correções de segurança.

Figura 8 – Exemplo *changelog* Debian



Fonte: <https://metadata.ftp-master.debian.org/changelogs/main/a/ant/ant_1.10.13-1_changelog> - Acessado em 18/08/2025

Em casos de erro de acesso ao *changelog* via URL, o código insere no banco de dados a *string* ***Check Manually***. A coleta foi automatizada através do código desenvolvido para este trabalho (arquivo api_debian_5.py).

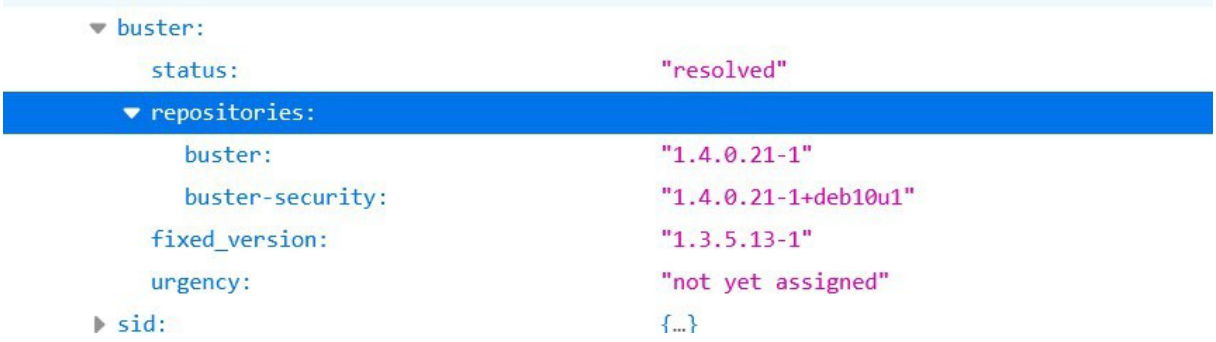
Debian apresentou uma das piores experiências para coleta de dados. Os principais desafios técnicos enfrentados nesta etapa foram a necessidade de descobrir os padrões de URLs que contêm os *changelogs*, a necessidade de acessar diversas URLs efetuando a técnica *web scraping*, gerando problemas de acesso e aumento significativo no tempo de coleta, fora os problemas com restrição de taxa de consulta (*rate limiting*) e a ocorrência de *timeouts* impostas pelos servidores HTTP.

Através deste processo, foram coletados e processados um total de **63.518** registros de CVEs do Debian, publicados durante o período estudado e armazenados na tabela **debian** do banco de dados. Com exceção da data de descoberta, data de publicação e da data de resolução no arquivo JSON (não fornecido pelo Debian), todos os demais campos exigidos pelo método ColATeC estão presentes nos registros.

Durante a implementação do método ColATeC, observaram-se inconsistências temporais na versão Debian Buster, pois ela apresentava desempenho significativamente superior

ao das demais versões. Essa discrepância foi atribuída ao fato de o campo de versão de correção exibir referências a versões descontinuadas, conforme exemplificado na figura 9. O problema, reportado à equipe do Debian (39) via Internet Relay Chat (IRC), foi confirmado como decorrente do término do ciclo de vida da versão Buster. Em nova coleta realizada em janeiro de 2025, os registros referentes a essa versão haviam sido removidos, justificando sua exclusão da análise no capítulo 5.

Figura 9 – Exemplo de vulnerabilidade no Debian Buster com versões conflitantes



Fonte: <https://security-tracker.debian.org/tracker/data/json> - Acessado em 05/07/2024

Apesar das estratégias adotadas para superar as dificuldades na coleta de informações, alguns dados relativos às correções não puderam ser recuperados. Por meio de consultas estruturadas ao banco de dados, foram obtidos os resultados exibidos na tabela 4, que apresenta a quantidade de CVEs resolvidas não registradas devido à ausência de dados sobre suas datas de resolução. As consultas utilizadas são detalhadas a seguir:

- **Total de CVEs resolvidas no Debian:**

  ```
  select count(CVE) from debian where Status like 'resolved';
  ```

- **Total de CVEs resolvidas no Debian sem a versão Buster:**

  ```
  select count(CVE) from debian where Status like 'resolved' and distro
  != 'buster';
  ```

- **Total de CVEs marcadas com *Check Manually* no Debian:**

  ```
  select count(CVE) from debian where Resolved = 'Check Manually'
  and Status = 'resolved';
  ```

- **Total de CVEs marcadas com *Check Manually* no Debian sem a versão buster:**

  ```
  select count(CVE) from debian where Resolved = 'Check Manually' and
  Status = 'resolved' and distro != 'buster';
  ```

Tabela 4 – Porcentagem de Correções não Calculadas no Debian

Versão Buster	Resolvidos	Check Manually	% Não Avaliada
Com	54.090	16.694	30,86%
Sem	45.505	9.234	20,29%

Fonte: Produzido pelo autor

Essas proporções correspondem a 16.694 ocorrências marcadas como "*Check Manually*" em um universo de 54.090 registros resolvidos quando a versão Buster é considerada. Ao excluir essa versão, permanecem 9.234 verificações pendentes entre 45.505 correções válidas, o que reduz a taxa de lacunas para 20,29%.

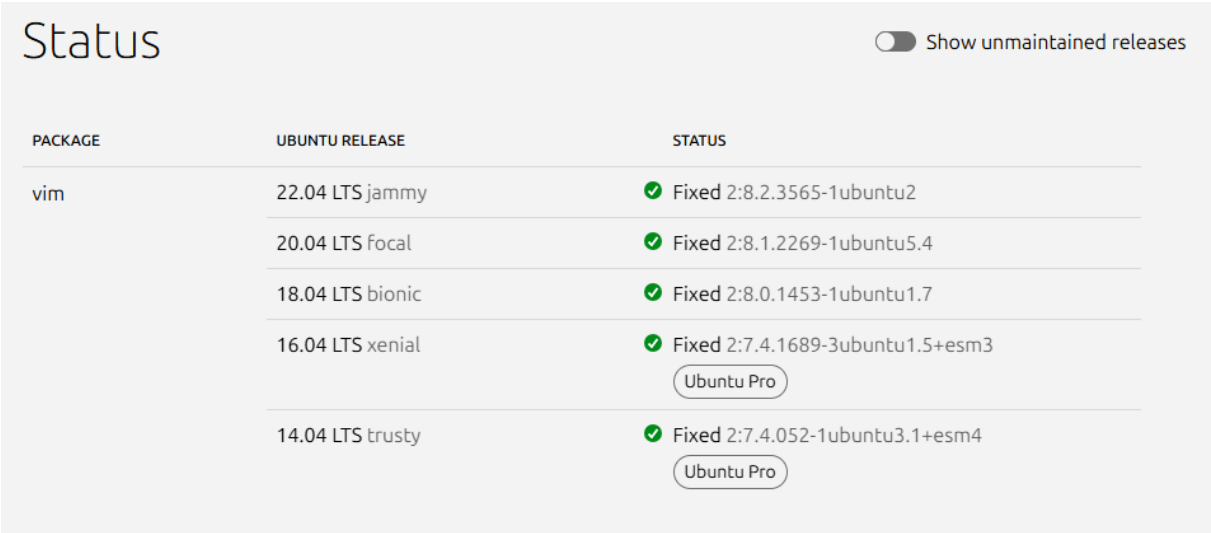
Apesar dessas lacunas, Debian mantém todo o *pipeline* de geração e publicação de dados acessível à comunidade por meio da Salsa (40). O código-fonte que produz os JSONs pode ser auditado, discutido e aprimorado através de *merge requests*, algo que não se observa na documentação das bases Red Hat, AlmaLinux ou Rocky, nas quais o canal para contribuições de código não é evidente. Essa abertura cria uma oportunidade concreta para que pesquisadores e profissionais proponham ajustes alinhados ao ColATeC, transformando críticas sobre dados incompletos em melhorias práticas. O desempenho médio, calculado com base na coleta de 100 registros, foi de 2,652737 segundos por registro.

4.6 Ubuntu

A distribuição Ubuntu disponibiliza informações de segurança (41) e detalhadas por meio do *Ubuntu Security Notices (USN)* (42), que registra vulnerabilidades, seu estado de correção e atualizações relacionadas. Para este trabalho, os dados foram obtidos do JSON oficial (43), acessado programaticamente via código desenvolvido para este trabalho (arquivo api_ubuntu_5.py).

A obtenção de dados completos exigiu desafios adicionais, pois Ubuntu não fornece diretamente as datas de resolução no JSON. Essas informações estão dispersas em *changelogs* públicos, acessíveis via URLs específicas. Observou-se ainda que o Ubuntu Pro (versão com suporte estendido) não divulga publicamente as datas de correções, limitando-se a indicar versões corrigidas com o sufixo **+esm**. Para exemplificar, considerou-se a **CVE-2021-3928**, ilustrada na figura 10, confirmando que as versões 16.04 e 14.04 foram corrigidas apenas no Ubuntu Pro.

Figura 10 – Informação sobre as versões dos pacotes Ubuntu e Ubuntu Pro



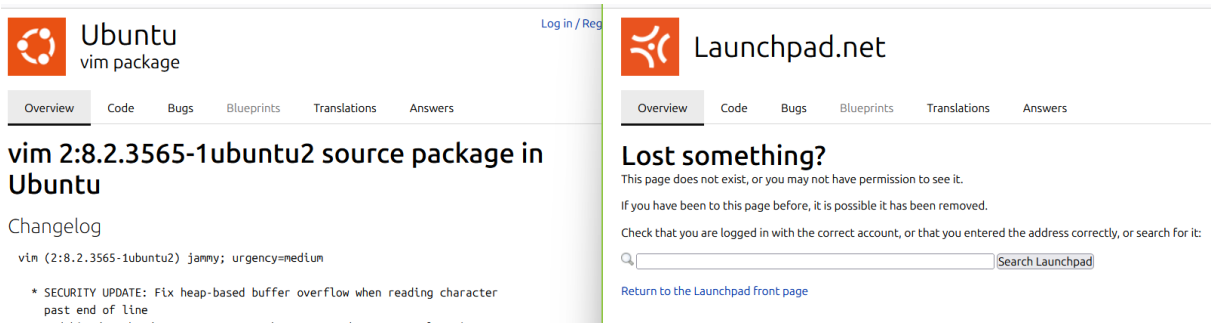
Fonte: Ubuntu CVE-2021-3928 (44)

Para obter o *changelog* remoto, identificou-se que o padrão da URL segue o formato: `https://launchpad.net/ubuntu/+source/package/support` onde:

- **`package:`** Nome do pacote Ubuntu associado à CVE;
- **`support:`** Versão na qual a vulnerabilidade foi corrigida.

A figura 11 compara os resultados da busca manual (simulando um processo automatizado) de *changelogs* da **CVE-2021-3928**. Enquanto a versão 20.04 exibe publicamente o *changelog* (lado esquerdo da figura, representando a URL <https://launchpad.net/ubuntu/+source/vim/2:8.2.3565-1ubuntu2>), a versão 16.04 Pro (lado direito da figura, representando a URL <https://launchpad.net/ubuntu/+source/vim/2:7.4.1689-3ubuntu1.5+esm3>) retorna uma mensagem de página indisponível. Em casos de erro no acesso ao *changelog*, o código insere no banco de dados a *string Check Manually*).

Figura 11 – Comparação CVE-2021-3928 entre Ubuntu e Ubuntu PRO



Fonte: Produzido pelo autor

Para contornar essa limitação, desenvolveu-se um *shell script* denominado ubuntu_pro.sh e o código ubuntu_pro.py que realizam as seguintes etapas:

- Leem vulnerabilidades marcadas com "`+esm`" que foram exportadas manualmente para um arquivo do tipo *Comma-Separated Values (CSV)* através da consulta SQL:
  `SELECT ubuntu.CVE, ubuntu.Distro,ubuntu.Package,ubuntu.Support FROM cvedb5.ubuntu where ubuntu.Support LIKE "%+esm%"`;
- Coletam dados de *changelogs* em instâncias *Virtual Machine (VM)* do Ubuntu Pro, utilizando assinatura gratuita para fins acadêmicos.

Problemas identificados no Ubuntu Pro:

- **Incompletude dos *Changelogs*:** O Ubuntu Pro não inclui todas as correções em seus *changelogs* públicos;
- **Volatilidade de Dados:** Alguns CVEs registrados anteriormente neste trabalho não estavam mais disponíveis em 2025, exigindo cruzamento com bases históricas.

Apesar das estratégias adotadas para superar as dificuldades na coleta de informações, alguns dados relativos às correções não puderam ser recuperados. Por meio de consultas estruturadas ao banco de dados, foram obtidos os resultados exibidos na tabela 5, que apresenta a quantidade de CVEs resolvidas não registradas devido à ausência de dados sobre suas datas de resolução. As consultas utilizadas são detalhadas a seguir:

- **Total de CVEs resolvidas no Ubuntu Padrão:**
  `select count(CVE) from ubuntu where Status like 'released';`
- **Total de CVEs resolvidas no Ubuntu Pro:**
  `select count(CVE) from ubuntupro where Status like 'released';`
- **Total de CVEs marcadas com *CHECK MANUALLY* no Ubuntu Padrão:**
  `select count(CVE) from ubuntu where Status like 'released' and Resolved like 'CHECK MANUALLY';`
- **Total de CVEs marcadas com *CHECK MANUALLY* no Ubuntu Pro:**
  `select count(CVE) from ubuntupro where Status like 'released' and Resolved like 'CHECK MANUALLY';`

Ubuntu apresentou a pior experiência para coleta de dados. Os principais desafios técnicos enfrentados nesta etapa foram a necessidade de descobrir os padrões de URLs que contêm os *changelogs*, a necessidade de acessar diversas URLs, gerando problemas de acesso e um aumento significativo no tempo de coleta.

Tabela 5 – Porcentagem de Correções não Calculadas no Ubuntu

Ubuntu	Resolvidos	CHECK MANUALLY	% Não Avaliada
Padrão	124.578	7.417	5,95%
Pro	124.578	6.638	5,33%

Fonte: Produzido pelo autor

No Ubuntu Pro, a coleta de dados foi muito pior devido à falta de *changelogs* públicos. Apesar disso, a replicação descrita neste capítulo permanece viável para qualquer pesquisador: a Canonical concede gratuitamente até cinco assinaturas Ubuntu Pro para uso pessoal, bastando solicitar o registro na plataforma.

Através deste processo, foram coletados e processados **175.685** registros de CVEs do Ubuntu, publicados durante o período estudado e armazenados na tabela **ubuntu** do banco de dados. O desempenho médio, calculado com base na coleta de 100 registros, foi de 2,20 segundos por registro.

Posteriormente, essa tabela foi replicada com o nome **ubuntupro** para receber os dados específicos do Ubuntu Pro. A execução dos *scripts* ubuntu_pro.sh e ubuntu_pro.py permitiu completar as informações referentes ao Ubuntu Pro na tabela clonada.

Com exceção do campo de data de descoberta, data de resolução no formato JSON (presente apenas localmente no Ubuntu Pro) e de uma maior facilidade no acesso aos dados, todos os demais campos exigidos pelo método ColATeC estão disponíveis nos registros do Ubuntu.

Em sintonia com o ecossistema Debian, a Canonical também expõe o código responsável pelo *feed* de vulnerabilidades no repositório ubuntu-com-security-api (45). Lá, contribuições são aceitas via *pull requests* e permitem que membros da comunidade proponham ajustes de estrutura ou novos campos que aproximem o formato atual dos requisitos do ColATeC. Diante da ausência de canais similares nas demais distribuições analisadas, Debian e Ubuntu tornam-se candidatos para esforços colaborativos que fortaleçam a qualidade e a padronização dos dados.

Para finalizar, responde-se à segunda questão de pesquisa deste trabalho (Q2: "*Como cada distribuição divulga suas vulnerabilidades de segurança?*") com base na exposição detalhada dos canais oficiais e metodologias de divulgação do NIST e das distribuições analisadas, apresentada ao longo deste capítulo.

Capítulo 5

Resultados e análise do estudo de caso

Este capítulo apresenta os resultados obtidos pela aplicação do método ColATeC, desenvolvidos em consonância com a metodologia detalhada no capítulo 3. O estudo gerou um extenso conjunto de dados sobre vulnerabilidades nas distribuições Linux investigadas, abrangendo o período definido na subseção 3.4.3. Os resultados aqui expostos visam responder diretamente às questões de pesquisa remanescentes, mediante a quantificação e comparação do desempenho das distribuições no que se refere à correção de vulnerabilidades.

A análise inicia-se com uma visão geral dos resultados, seguida pela comparação detalhada dos tempos de correção e finaliza com uma investigação aprofundada sobre o perfil das vulnerabilidades e o comportamento específico de cada distribuição. Os dados foram extraídos e processados por meio do código desenvolvido results.py.

5.1 Vulnerabilidades de pacotes comuns entre ecossistemas

Uma análise comparativa direta do tempo de correção entre todas as vulnerabilidades de cada distribuição pode introduzir um viés relacionado ao escopo de pacotes mantidos por cada ecossistema. Distribuições com repositórios de *software* mais extensos estão inerentemente expostas a um número maior e mais diversificado de vulnerabilidades. Para mitigar esse viés e estabelecer uma base de comparação equitativa, realizou-se uma análise focada nas vulnerabilidades presentes em pacotes de *software* comuns a ambos os

ecossistemas (RPM e DEB).

Neste trabalho, considerou-se uma vulnerabilidade como "comum" quando a CVE correspondente foi identificada em pelo menos uma distribuição do ecossistema RPM e, simultaneamente, em pelo menos uma do ecossistema DEB. Adotou-se como critério o registro do pacote envolvido na CVE para fins de comparação. Contudo, como o mesmo pacote pode receber denominações distintas em cada distribuição (por exemplo, **httpd** no Red Hat e **apache2** no Debian), tornou-se necessário realizar um processo de normalização da nomenclatura. Para isso, desenvolveu-se o código packages.py, que mapeia nomes variantes para um identificador canônico. A figura 6 ilustra um exemplo do resultado desse processo, apresentando tanto o nome original do pacote quanto sua versão padronizada no banco de dados.

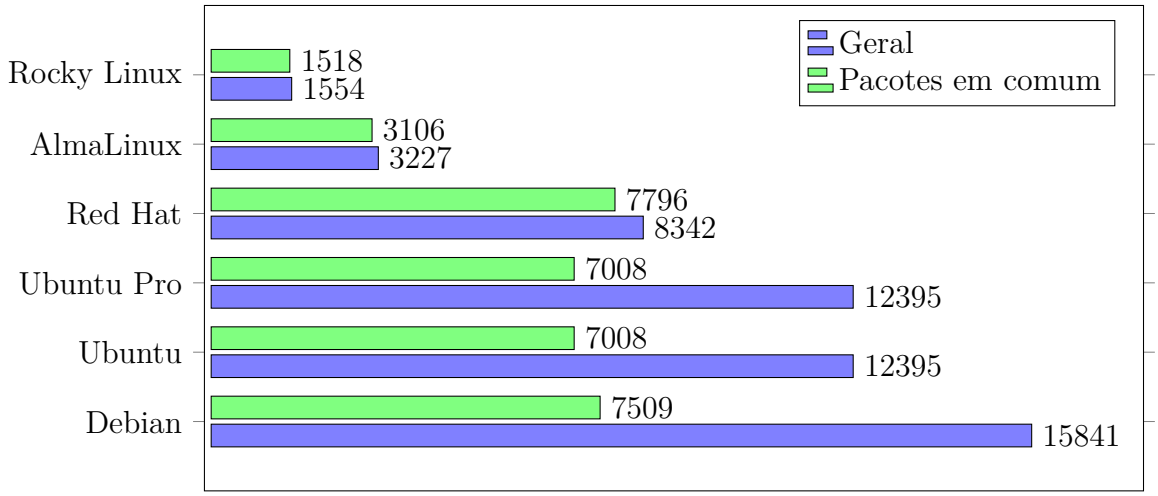
O objetivo do processo de normalização foi construir o conjunto final de **pacotes** que são comuns a ambos os ecossistemas (RPM e DEB), isto é, aqueles afetados por pelo menos uma CVE inicialmente identificada como comum. Uma vez extraídos os identificadores canônicos de todos os pacotes envolvidos nas CVEs comuns, realizou-se uma nova consulta à base de dados. Essa consulta buscou **todas** as vulnerabilidades (CVEs) associadas a esses pacotes comuns, mesmo que a CVE específica não tivesse sido corrigida em um dos ecossistemas. Dessa forma, foi possível construir com precisão o conjunto final de vulnerabilidades de pacotes comuns a serem analisadas, garantindo uma comparação válida entre os ecossistemas.

Esta abordagem supera a limitação identificada no trabalho de JANSON (9), no qual o mapeamento entre pacotes de diferentes distribuições era realizado por meio de uma lista definida manualmente, o que exigia uma listagem exaustiva de nomes alternativos. Este trabalho utiliza a ocorrência simultânea da CVE nos dois ecossistemas como critério primário para determinar quais pacotes são comuns, eliminando a necessidade desse mapeamento manual e assegurando uma identificação automática dos pacotes equivalentes com o suporte do método ColATeC.

5.2 Apresentação geral dos dados

Esta seção apresenta e analisa os resultados gerais obtidos, com detalhamentos específicos nas subseções subsequentes. A figura 12 apresenta o total de CVEs únicas (tanto do cenário geral quanto do subconjunto de pacotes em comum) que foram identificadas ao longo do período de coleta de cinco anos estabelecido.

Figura 12 – Total de CVEs Únicas no Período de 5 Anos



Fonte: Produzido pelo autor

A análise dos dados expostos na figura 12 permite extrair as seguintes conclusões:

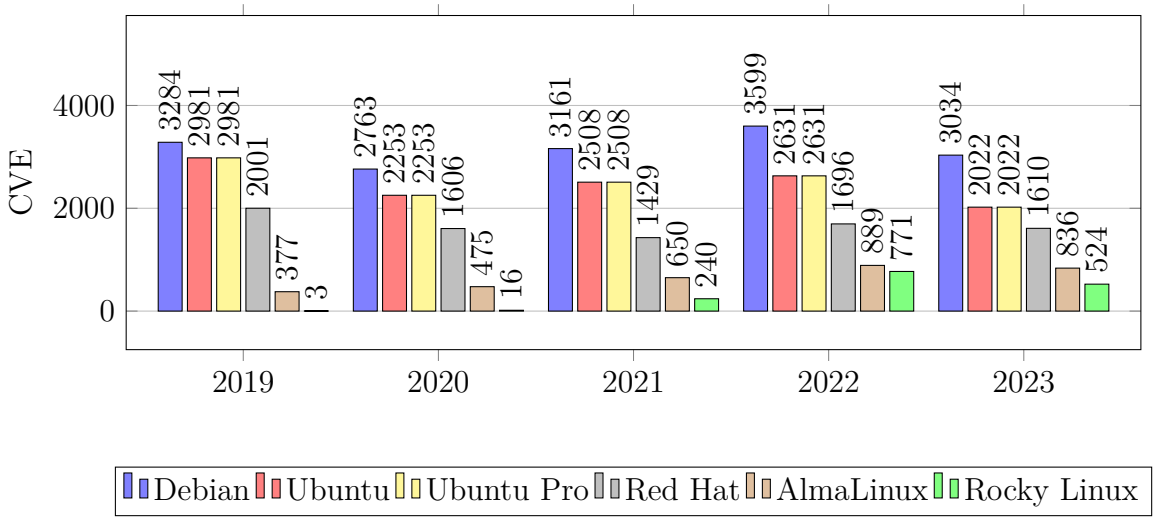
i) **Vulnerabilidades no Cenário Geral**: O número mais elevado de CVEs nas distribuições baseadas em Debian (Debian e Ubuntu) é um reflexo direto de sua filosofia de gerenciamento de pacotes e da abrangência de seus repositórios, que oferecem uma variedade de *software* significativamente maior. Este dado, portanto, não indica menor segurança, mas sim uma maior superfície de exposição. A eficácia da segurança reside na capacidade de mitigar e corrigir falhas de forma ágil, e não apenas na contagem bruta de vulnerabilidades, que está intrinsecamente ligada à arquitetura e ao escopo dos repositórios;

ii) **Vulnerabilidades nos Pacotes em Comum**: Ao restringir a análise ao subconjunto de pacotes comuns, o cenário se inverte. A Red Hat passa a liderar com 7.796 CVEs catalogadas, superando Debian (7.509) e Ubuntu (7.008). Este resultado não sugere uma maior fragilidade da Red Hat, mas sim o oposto: evidencia um processo de monitoramento e catalogação de vulnerabilidades mais maduro e proativo. Neste contexto, um número maior de CVEs registradas indica que a equipe de segurança da distribuição é mais eficiente em identificar, reconhecer e tornar pública a existência de falhas, o que constitui o primeiro e mais crucial passo para a sua correção. Portanto, para um mesmo conjunto de *software*, uma maior quantidade de vulnerabilidades **reportadas** é um indicador positivo da maturidade da gestão de segurança.

A mesma lógica se aplica, de forma ainda mais acentuada, à análise das distribuições AlmaLinux e Rocky Linux. O número substancialmente inferior de CVEs (3.227 e 1.554, respectivamente) não deve ser interpretado como um sinal de segurança superior. Embora parte dessa disparidade possa ser atribuída ao ciclo de vida mais curto de ambas as

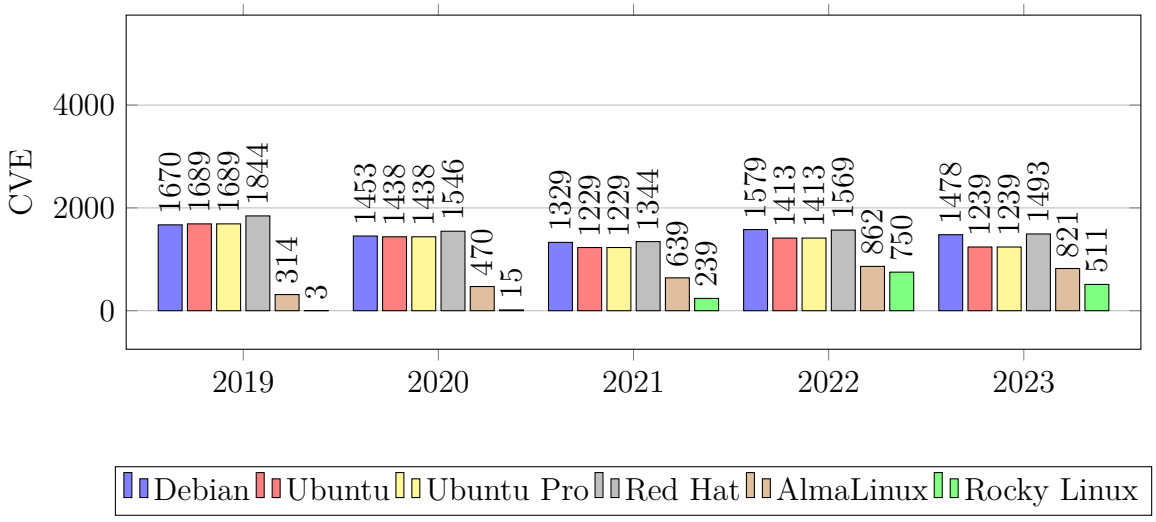
distribuições — criadas entre o final de 2020 e início de 2021 como resposta às mudanças no CentOS (46, 47, 48) —, a principal inferência é que essa contagem reduzida evidencia um processo de tratamento de vulnerabilidades ainda incipiente. A discrepância sugere que suas equipes de segurança e processos de automação para identificação e registro de CVEs ainda não atingiram o mesmo nível de maturidade das distribuições mais estabelecidas. Assim, os dados indicam um sub-relatório de vulnerabilidades em comparação com as distribuições com ecossistemas de segurança mais desenvolvidos. A figura 13 e a figura 14 aprofundam essa análise ao apresentar a evolução anual das CVEs.

Figura 13 – Total de CVEs Únicas Ano a Ano - Geral



Fonte: Produzido pelo autor

Figura 14 – Total de CVEs Únicas Ano a Ano - Pacotes Comuns



Fonte: Produzido pelo autor

A análise dos dados apresentados nas figuras 13 e 14 permite inferir as seguintes conclusões complementares:

i) **Vulnerabilidades em geral**: A análise dos dados evidencia que, em termos quantitativos, a distribuição Debian apresenta consistentemente o maior volume de vulnerabilidades no cenário geral. Conclui-se que, de modo geral, o Debian possui a maior quantidade de vulnerabilidades reportadas, provavelmente refletindo seu repositório de software mais extenso;

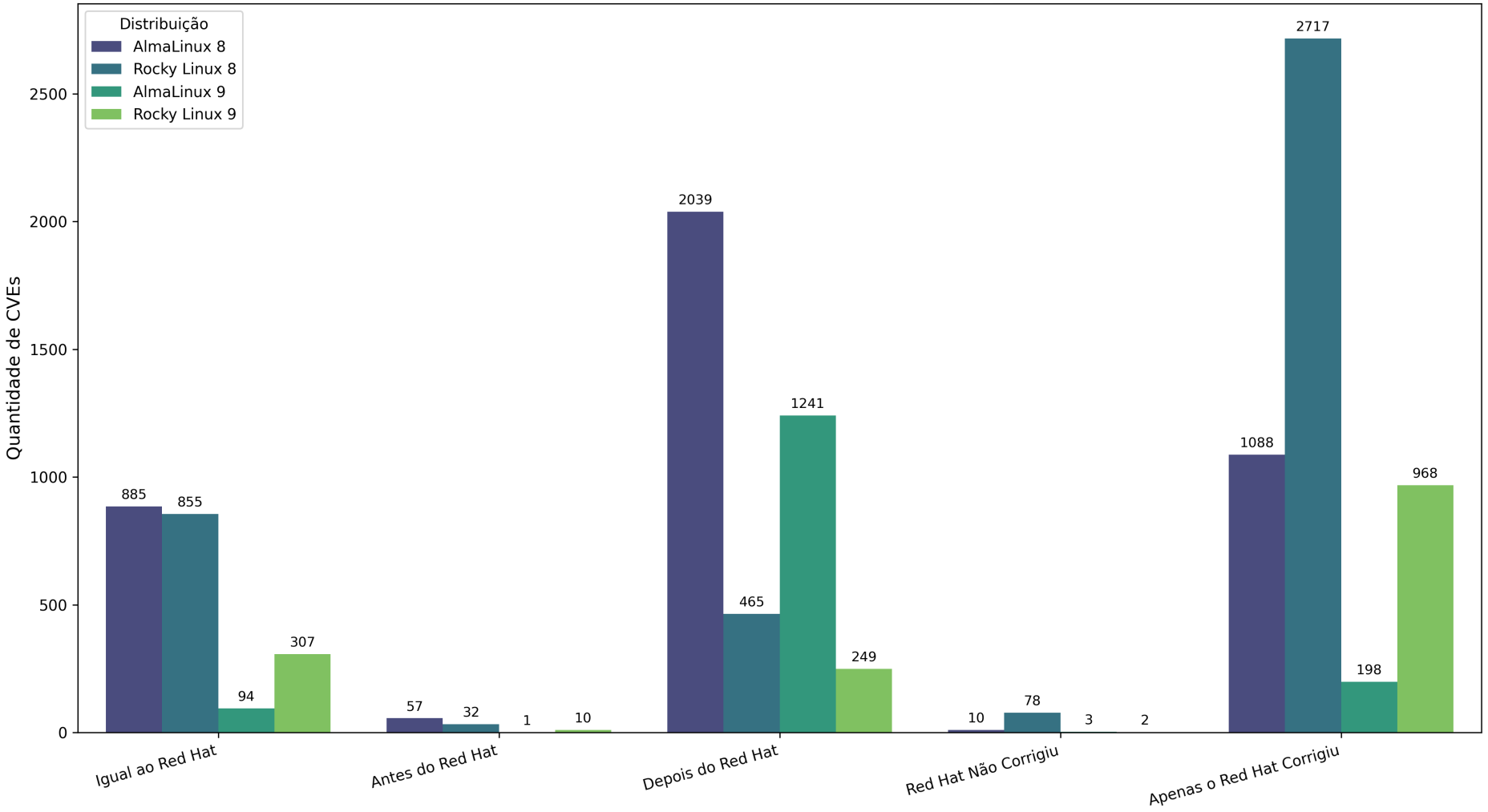
ii) **Vulnerabilidades nos pacotes em comum**: Ao analisar o subconjunto de vulnerabilidades presentes em pacotes comuns a ambos os ecossistemas, observa-se uma variação na distribuição que apresenta a maior quantidade. Por exemplo, em 2019 o Red Hat registrou a maior quantidade de vulnerabilidades de pacotes em comum, enquanto em 2022 foi o Debian. Nota-se uma proximidade nos valores entre Debian, Ubuntu e Red Hat, com exceção das distribuições Rocky Linux e AlmaLinux, que novamente apresentam uma quantidade significativamente menor de vulnerabilidades;

iii) **Número persistentemente baixo de CVEs no AlmaLinux e Rocky Linux**: Mesmo após o período inicial de estabilização, ambas as distribuições mantiveram quantidades significativamente inferiores às demais analisadas. Este comportamento motivou uma investigação mais aprofundada, conforme detalhado na subseção 5.2.1.

5.2.1 Análise mais detalhada sobre AlmaLinux e Rocky Linux

Diante do comportamento atípico relacionado à baixa quantidade de CVEs registradas para as distribuições AlmaLinux e Rocky Linux, realizou-se uma análise mais aprofundada para investigar as possíveis causas dessa discrepância. O fato de esses valores permanecerem significativamente inferiores mesmo quando comparados aos do Red Hat nos anos de 2022 e 2023, após o período inicial de estabilização dos projetos, indica a necessidade de uma investigação específica sobre os mecanismos de rastreamento, reportamento e correção adotados por essas distribuições. Para tanto, por meio do código desenvolvido results.py, foram extraídas as seguintes informações apresentadas pelas figuras a seguir.

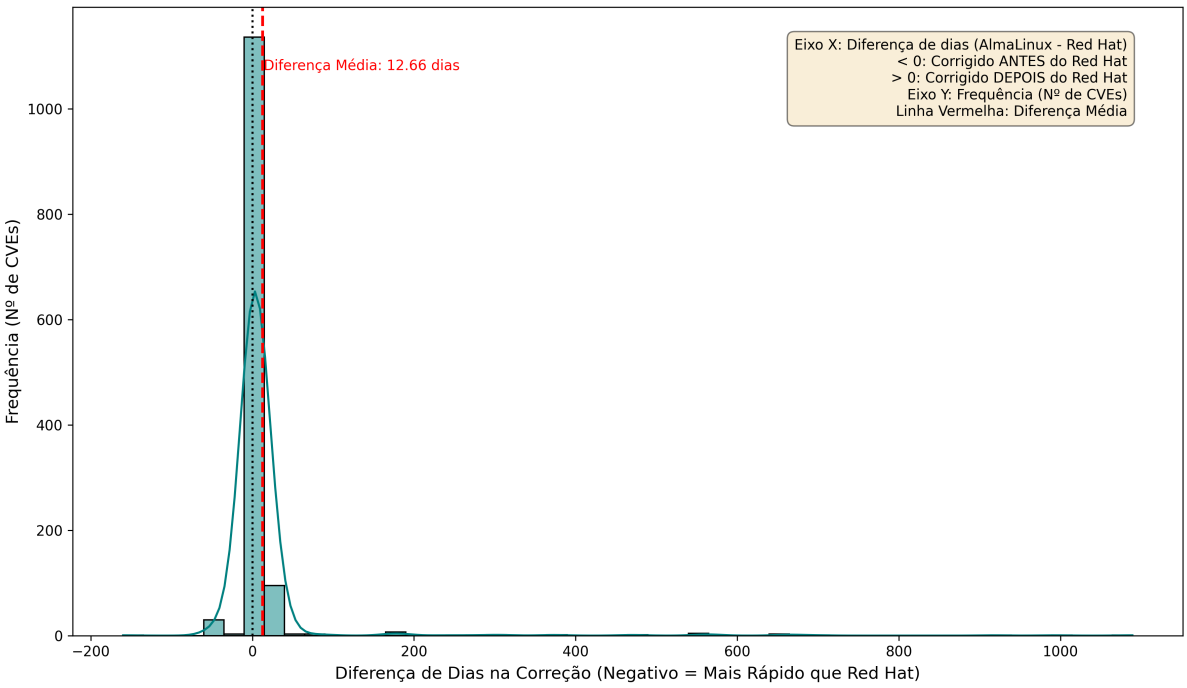
Figura 15 – Comparação de Correções AlmaLinux e Rocky Linux vs. Red Hat



Fonte: Produzido pelo autor

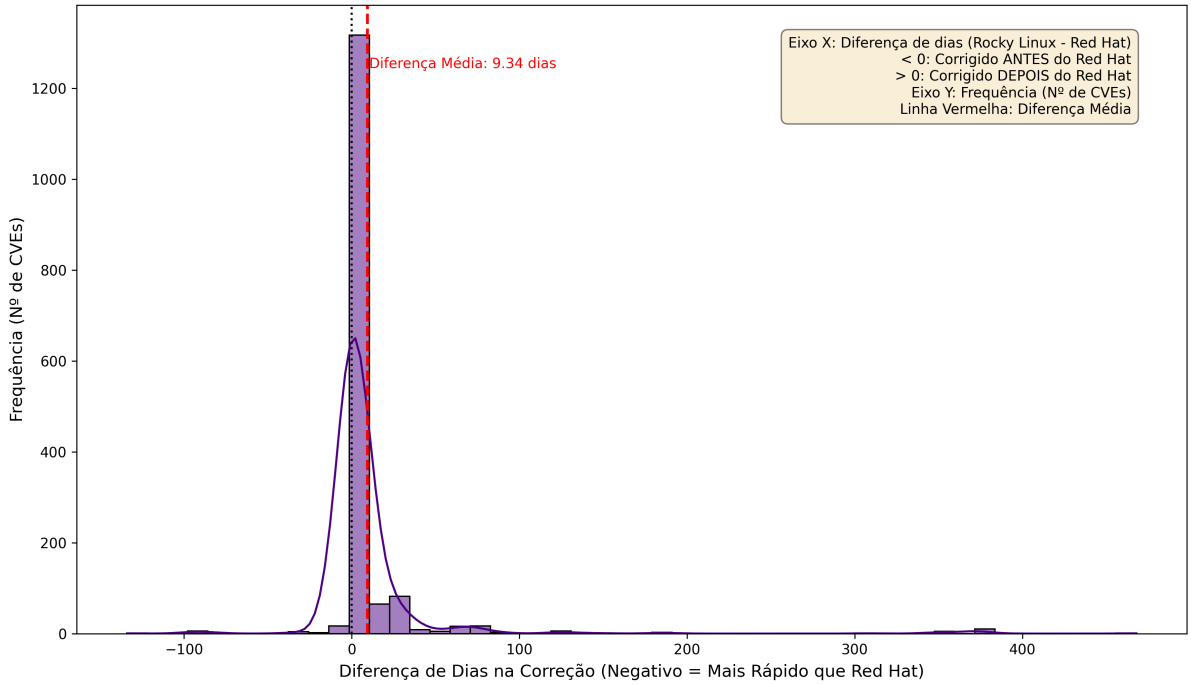
A figura 15 evidencia um aspecto relevante: embora as distribuições AlmaLinux e Rocky Linux busquem acompanhar o ritmo de correções do Red Hat, elas não corrigem integralmente todas as vulnerabilidades reportadas. Este comportamento explica a quantidade significativamente inferior de CVEs registradas para essas distribuições em comparação com as demais analisadas.

Figura 16 – Histograma de Diferença de Dias: AlmaLinux vs. Red Hat



Fonte: Produzido pelo autor

Figura 17 – Histograma de Diferença de Dias: Rocky Linux vs. Red Hat



Fonte: Produzido pelo autor

As figuras 16 e 17 evidenciam que, embora AlmaLinux e Rocky Linux não corrijam todas as vulnerabilidades presentes no Red Hat, aquelas que são corrigidas por essas distribuições tendem a acompanhar o tempo de correção da distribuição Red Hat, com poucas exceções.

Ao final desta seção, é possível responder a Q3: "*Qual distribuição apresenta o maior número de falhas de segurança reportadas?*". A distribuição Linux Debian é a que apresenta mais vulnerabilidades no cenário geral (15.841 CVEs). Cabe ressaltar que, no cenário de pacotes em comum, o Red Hat apresenta a maior quantidade (7.796 CVEs), conforme apresentado na figura 12.

5.3 Estado de correção das vulnerabilidades

Concluída a análise geral das vulnerabilidades, realizou-se a comparação do estado de correção entre as distribuições para avaliar o desempenho. As tabelas 6 e 7 apresentam o estado de correção das vulnerabilidades em cada distribuição. Para esta análise, foram consideradas todas as ocorrências de vulnerabilidades, incluindo aquelas repetidas em uma mesma distribuição devido à presença em múltiplos pacotes ou versões diferentes.

Tabela 6 – Tabela Geral de Distribuição de Correções

Distribuição	Corrigidas (>0 dias)	Corrigidas (Dia Zero)	Não Corrigidas	Datas Negativas	Total Processado
AlmaLinux	6.554 (99.6%)	26 (0.4%)	0 (0.0%)	0 (0.0%)	6.580
Debian	35.093 (68.8%)	38 (0.1%)	14.715 (28.9%)	1.140 (2.2%)	50 986
Red Hat	23.319 (40.0%)	379 (0.7%)	33.883 (58.1%)	716 (1.2%)	58.297
Rocky Linux	2.860 (99.4%)	4 (0.1%)	0 (0.0%)	13 (0.5%)	2.877
Ubuntu	38.335 (59.2%)	690 (1.1%)	17.530 (27.1%)	8.205 (12.7%)	64.760
Ubuntu Pro	39.349 (60.8%)	697 (1.1%)	16.438 (25.4%)	8.276 (12.8%)	64.760

Fonte: Produzido pelo autor

Tabela 7 – Tabela de Distribuição de Correções - Pacotes Comuns

Distribuição	Corrigidas (>0 dias)	Corrigidas (Dia Zero)	Não Corrigidas	Datas Negativas	Total Processado
AlmaLinux	6.531 (99.6%)	26 (0.4%)	0 (0.0%)	0 (0.0%)	6.557
Debian	25.033 (71.1%)	19 (0.1%)	9.711 (27.6%)	423 (1.2%)	35.186
Red Hat	23.277 (40.0%)	379 (0.7%)	33.799 (58.1%)	714 (1.2%)	58.169
Rocky Linux	2.854 (99.4%)	4 (0.1%)	0 (0.0%)	13 (0.5%)	2.871
Ubuntu	37.353 (62.9%)	668 (1.1%)	14.299 (24.1%)	7.049 (11.9%)	59.369
Ubuntu Pro	38.199 (64.3%)	675 (1.1%)	13.389 (22.6%)	7.106 (12.0%)	59.369

Fonte: Produzido pelo autor

A análise comparativa dos dados apresentados nas tabelas anteriores (que detalham a distribuição de correções para o conjunto total de pacotes e para o subconjunto de pacotes comuns) permite inferir as seguintes conclusões:

i) **Consistência entre cenários**: Observa-se uma notável similaridade nos percentuais de cada categoria de correção ao se comparar os dados do total de pacotes com os dos pacotes comuns. Essa baixa variação sugere que o comportamento de correção das distribuições é consistente, indicando que as políticas de segurança são aplicadas de maneira homogênea na maior parte de seu ecossistema de software;

ii) **Desempenho de distribuições derivadas (AlmaLinux e Rocky Linux)**: As distribuições AlmaLinux e Rocky Linux, por serem derivadas diretamente do Red Hat (RHEL), apresentam um perfil distinto. O número absoluto de vulnerabilidades processadas é significativamente menor que o do RHEL, e a taxa de vulnerabilidades não corrigidas é nula (0,0%). Este fato, longe de indicar uma segurança inerentemente superior, sugere fortemente uma metodologia de divulgação diferente, provavelmente na qual as vulnerabilidades só são contabilizadas após a correção ser disponibilizada pela RHEL e incorporada pela derivada;

iii) **Alta taxa de vulnerabilidades não corrigidas na Red Hat**: Um aspecto que merece destaque é a elevada proporção de vulnerabilidades classificadas como "Não Corrigidas" no Red Hat (58,1%). Este resultado requer uma investigação mais aprofundada, a ser realizada na seção 5.5;

iv) **Correções no dia zero são minoritárias**: Em todas as distribuições, a porcentagem de vulnerabilidades corrigidas no mesmo dia de sua publicação ("Dia Zero") é baixa, variando de 0,1% a 1,1%. Isto corrobora a premissa de que o processo de correção, envolvendo triagem, desenvolvimento, teste e distribuição da correção, geralmente exige um período superior a 24 horas, sendo as correções imediatas uma exceção, provavelmente reservada a falhas críticas;

v) **Desempenho do Ubuntu Pro**: O Ubuntu Pro, versão com suporte estendido e *patches* de segurança adicionais, apresenta desempenho ligeiramente superior ao Ubuntu padrão, com maior percentual de vulnerabilidades corrigidas e menor de não corrigidas. Esta diferença, embora pequena (na ordem de 1-3 pontos percentuais), está alinhada com as expectativas divulgadas pela própria Canonical (49). Na seção 5.6 será apresentado análise da criticidade das vulnerabilidades corrigidas apenas no Ubuntu Pro;

vi) **Inconsistências nos dados (datas negativas)**: A presença de registros com "Datas Negativas" (situações em que a data de correção T_1 é anterior à data de publicação da vulnerabilidade T_0), embora represente uma minoria dos casos (com maior incidência no Ubuntu, aproximadamente 12%), indica inconsistências no processo de registro ou divulgação das informações. Tais inconsistências introduzem ruído na análise e apontam para desafios relacionados à confiabilidade e transparência das fontes de dados de segurança, aspectos fundamentais para a precisão de estudos comparativos.

Registros marcados como *Not Affected* foram excluídos de todas as estatísticas temporais discutidas neste capítulo. Esses eventos permanecem catalogados no banco **cvedb5**, mas não influenciam médias, medianas ou distribuições de tempo, garantindo que apenas correções efetivamente publicadas sejam consideradas na comparação entre distribuições.

5.3.1 Criticidade das vulnerabilidades

A avaliação da gravidade das vulnerabilidades constitui uma etapa crucial para elucidar as políticas de priorização empregadas pelos mantenedores das distribuições. A categorização das falhas corrigidas permite inferir quais níveis de risco recebem tratamento prioritário e o respectivo tempo de resposta, refletindo diretamente na postura de segurança de cada ecossistema. Diante disso, realizou-se uma análise comparativa inicial entre as classificações de severidade atribuídas pelo NIST e aquelas reportadas por cada distribuição.

Tabela 8 – Comparações de severidade entre as distribuições e o NIST - Geral

NIST / Red Hat	**NIST**	**Red Hat**
Critical / Critical	601	74
High / Important	3.461	1.522
Medium / Moderate	3.869	5.023
Low / Low	311	1.609
[ ] / None	100	114
Total	**8.342**	**8.342**

(a) NIST x Red Hat

NIST / Alma	**NIST**	**Alma**
Critical / Critical	199	23
High / Important	1.238	1.225
Medium / Moderate	1.658	1.997
Low / Low	122	146
[ ] / None	10	0
Total	**3.227**	**3.391**

(b) NIST x AlmaLinux

NIST / Debian	**NIST**	**Debian**
Critical / Critical	1.565	0
High / High	6.750	0
Medium / Medium	7.086	4
Low / Low	439	410
– / Unimportant	0	4.025
– / end-of-life	0	1.106
[ ] / not yet assigned	1	12.720
Total	**15.841**	**18.265**

(c) NIST x Debian

NIST / Ubuntu	**NIST**	**Ubuntu**
Critical / Critical	1.178	1
High / High	5.283	264
Medium / Medium	5.561	9.995
Low / Low	373	1.912
– / Negligible	0	166
[ ] / unknown	0	57
Total	**12.395**	**12.395**

(d) NIST x Ubuntu

NIST / Rocky	**NIST**	**Rocky**
Critical / Critical	104	22
High / Important	687	693
Medium / Moderate	714	883
Low / Low	46	34
[ ] / None	3	0
Total	**1.554**	**1.632**

(e) NIST x Rocky Linux

Fonte: Produzido pelo autor

Tabela 9 – Comparações de severidade entre as distribuições e o NIST - Pacotes em Comum

NIST / Red Hat	**NIST**	**Red Hat**
Critical / Critical	559	64
High / Important	3.293	1.415
Medium / Moderate	3.643	4.702
Low / Low	300	1.556
[ ] / None	1	59
Total	**7.796**	**7.796**

(a) NIST x Red Hat

NIST / Alma	**NIST**	**Alma**
Critical / Critical	195	22
High / Important	1.216	1.136
Medium / Moderate	1.578	1.962
Low / Low	117	143
[ ] / None	0	0
Total	**3.106**	**3.263**

(b) NIST x AlmaLinux

NIST / Debian	**NIST**	**Debian**
Critical / Critical	560	0
High / High	3.222	0
Medium / Medium	3.438	0
Low / Low	288	176
– / Unimportant	0	1.763
– / end-of-life	0	39
[ ] / not yet assigned	1	6.449
Total	**7.509**	**8.427**

(c) NIST x Debian

NIST / Ubuntu	**NIST**	**Ubuntu**
Critical / Critical	491	0
High / High	2.946	208
Medium / Medium	3.296	5.403
Low / Low	274	1.258
– / Negligible	0	118
[ ] / unknown	1	21
Total	**7.008**	**7.008**

(d) NIST x Ubuntu

NIST / Rocky Linux	**NIST**	**Rocky Linux**
Critical / Critical	99	21
High / Important	671	669
Medium / Moderate	703	869
Low / Low	45	33
[ ] / None	0	0
Total	**1.518**	**1.592**

(e) NIST x Rocky Linux

Fonte: Produzido pelo autor

Conforme evidenciado pelas tabelas 8 e 9, observam-se divergências significativas na avaliação da criticidade de vulnerabilidades. Este comportamento corrobora os achados de WU et al. (13), que apresentaram que a severidade de vulnerabilidades pode variar conforme versões e distribuições distintas. A seguir, apresentam-se brevemente os critérios utilizados pelo NIST e por cada distribuição para classificação de severidade.

i) O NIST adota o cenário mais adverso para vulnerabilidades com informações limitadas (50);

ii) Distribuições como Red Hat (51) e Ubuntu (52) adaptam o CVSS conforme configurações específicas (ex.: execução de serviços com privilégios reduzidos ao invés de root);

iii) O Debian classifica vulnerabilidades com base no impacto direto em seus pacotes (53);

iv) AlmaLinux e Rocky Linux não detalham publicamente seus critérios de classificação.

A análise das tabelas 8 e 9 revelou que o número total de vulnerabilidades em algumas distribuições excedeu a contagem documentada na figura 12. Essa variação ocorre pelo fato de uma mesma CVE poder ser categorizada com diferentes níveis de severidade em versões distintas de uma mesma distribuição, como exemplificado na tabela 10, reforçando novamente os achados de WU et al. (13). O método ColATeC apresentou robustez ao lidar com essas variações, validando sua eficácia para análises comparativas.

Tabela 10 – CVE com severidades diferentes na mesma distribuição

CVE	**Distribuição e Versão**	**Severidade**
CVE-2021-28153	Almalinux 8[1]	moderate
CVE-2021-28153	Almalinux 9[2]	low

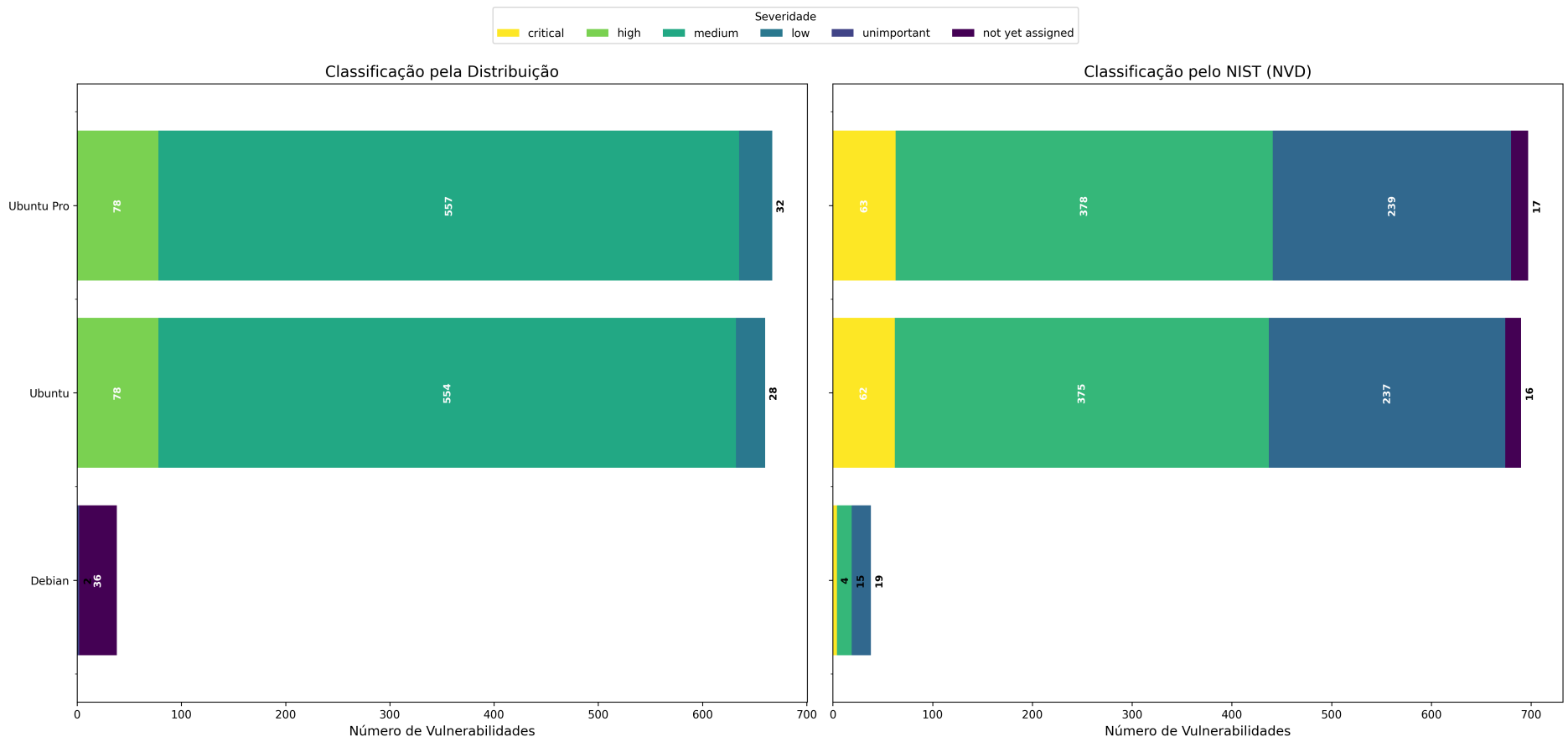
Fonte: Produzido pelo autor

Para complementar a análise das vulnerabilidades, as figuras 18, 19, 20 e 21 apresentam a distribuição das classificações de criticidade atribuídas pelo NIST e pelas distribuições, considerando os seguintes cenários:

- **Vulnerabilidades corrigidas no dia da publicação (Dia Zero)**: Identifica os tipos de vulnerabilidades que recebem correção imediata;
- **Vulnerabilidades corrigidas após a publicação (>0 Dias)**: Apresenta a criticidade das vulnerabilidades que não foram remediadas no dia de sua divulgação.

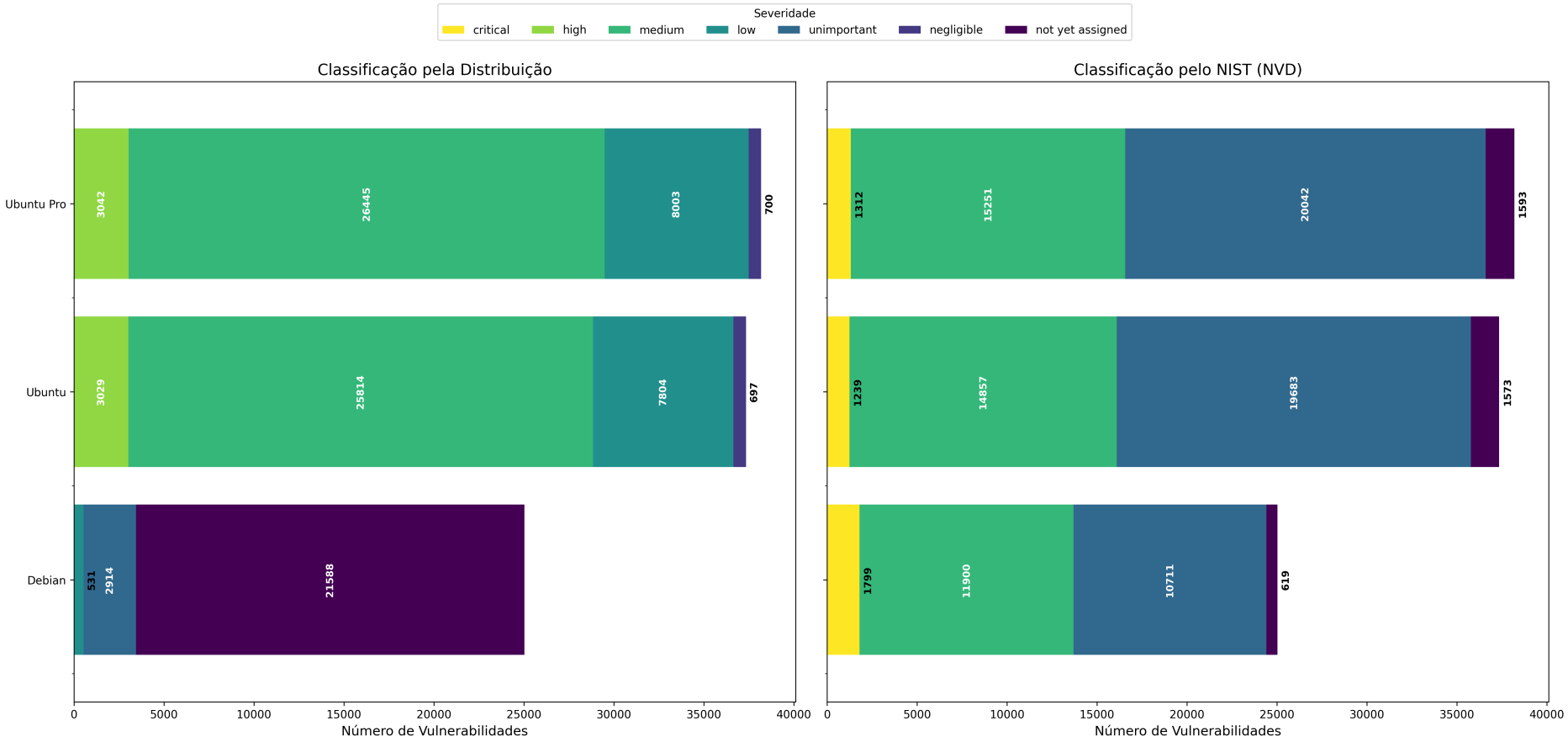
[1] <https://errata.almalinux.org/8/ALSA-2021-4385.html> (Acessado em 27/10/2024)
[2] <https://errata.almalinux.org/9/ALSA-2022-8418.html> (Acessado em 27/10/2024)

Figura 18 – Comparação de Severidades de CVEs Corrigidas Dia Zero - Família .deb



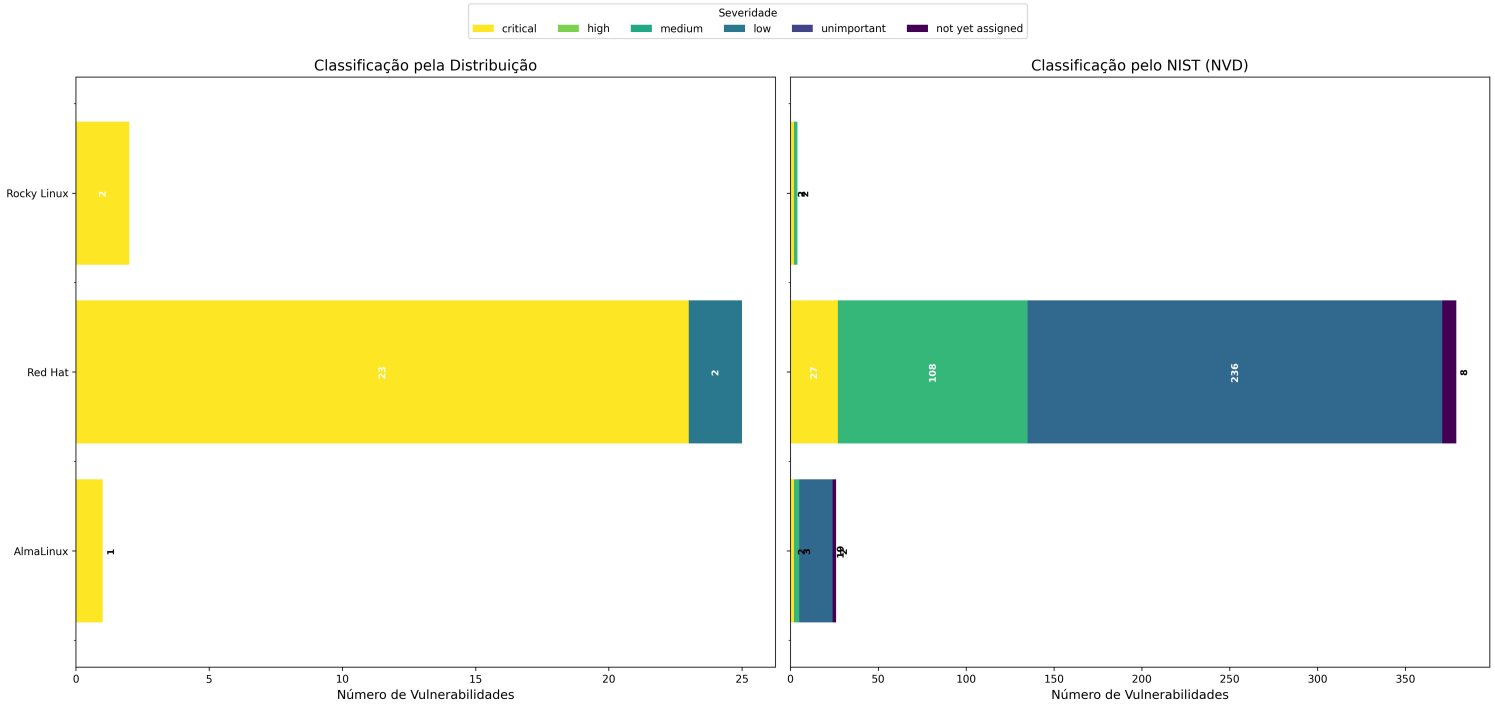
Fonte: Produzido pelo autor

Figura 19 – Comparação de Severidades de CVEs Corrigidas Maior que Zero Dias - Família .deb



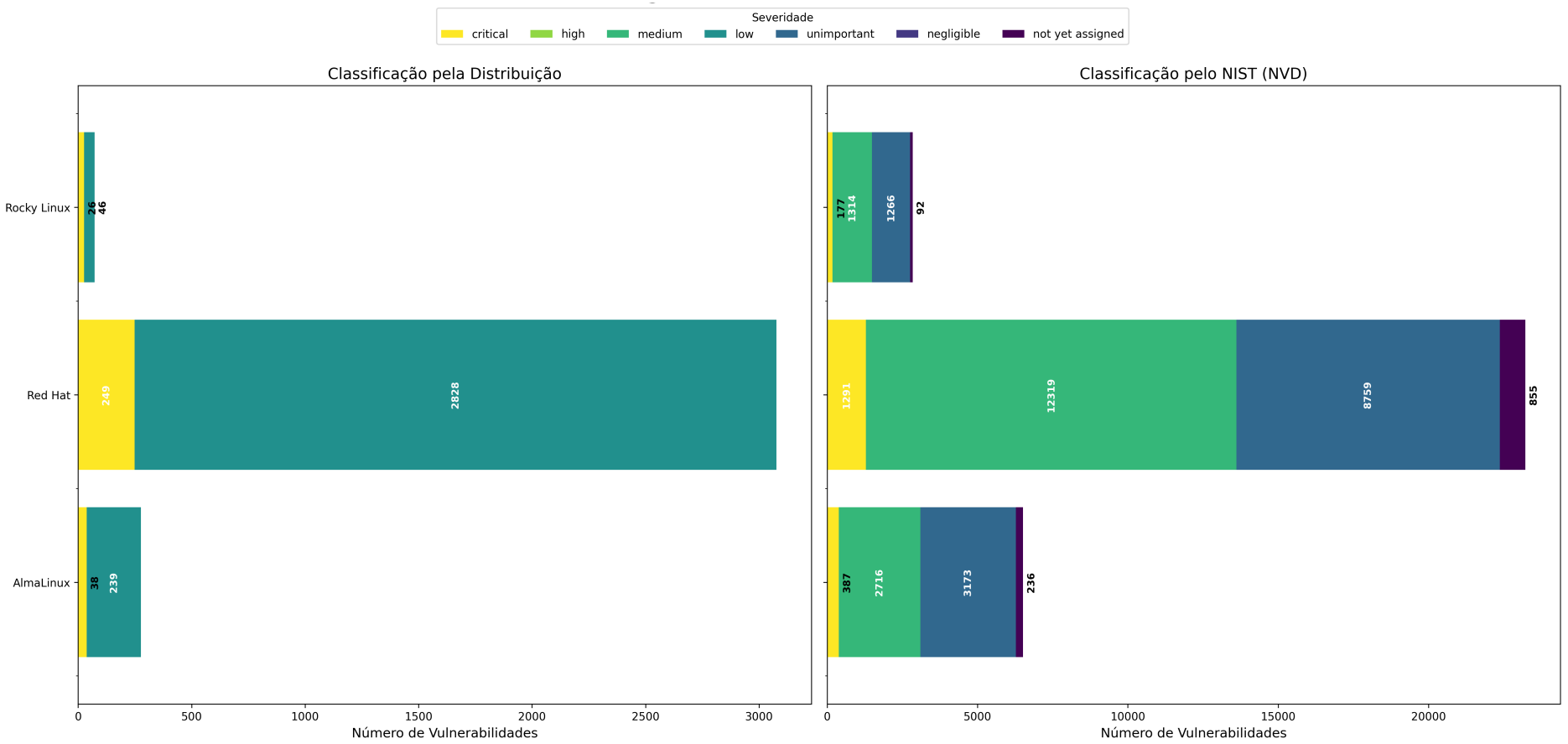
Fonte: Produzido pelo autor

Figura 20 – Comparação de Severidades de CVEs Corrigidas Dia Zero - Família .rpm



Fonte: Produzido pelo autor

Figura 21 – Comparação de Severidades de CVEs Corrigidas Maior que Zero Dias - Família .rpm



Fonte: Produzido pelo autor

A observação das figuras em análise revela um padrão consistente e alinhado às boas práticas de segurança: a grande maioria das correções de dia zero concentra-se nas vulnerabilidades de severidade *High* (Alta) e *Medium* (Média). Esse comportamento é esperado, uma vez que tais categorias representam falhas com potencial de exploração elevado, capazes de comprometer a confidencialidade, a integridade ou a disponibilidade dos sistemas. Ressalta-se, contudo, a classificação utilizada pelo Debian, que apresenta grande parte de suas vulnerabilidades como *not yet assigned* (não atribuída), indicando que a distribuição não divulga a criticidade em um primeiro momento, o que dificulta análises automatizadas e a priorização com base em risco. A análise da criticidade das vulnerabilidades não corrigidas e daquelas que apresentam datas negativas será realizada na seção 5.4.

5.4 Vulnerabilidades não corrigidas e com datas negativas

As vulnerabilidades não corrigidas e com datas negativas foram analisadas separadamente e serão apresentadas a seguir. Conforme evidenciado pelas tabelas 6 e 7, uma análise superficial poderia levar à conclusão equivocada de que AlmaLinux e Rocky Linux possuem desempenho superior na correção de vulnerabilidades, dada a porcentagem significativamente maior de correções realizadas. Contudo, é crucial considerar a existência de valores nulos de vulnerabilidades não corrigidas em conjunto com o volume total reduzido de entradas processadas por essas distribuições.

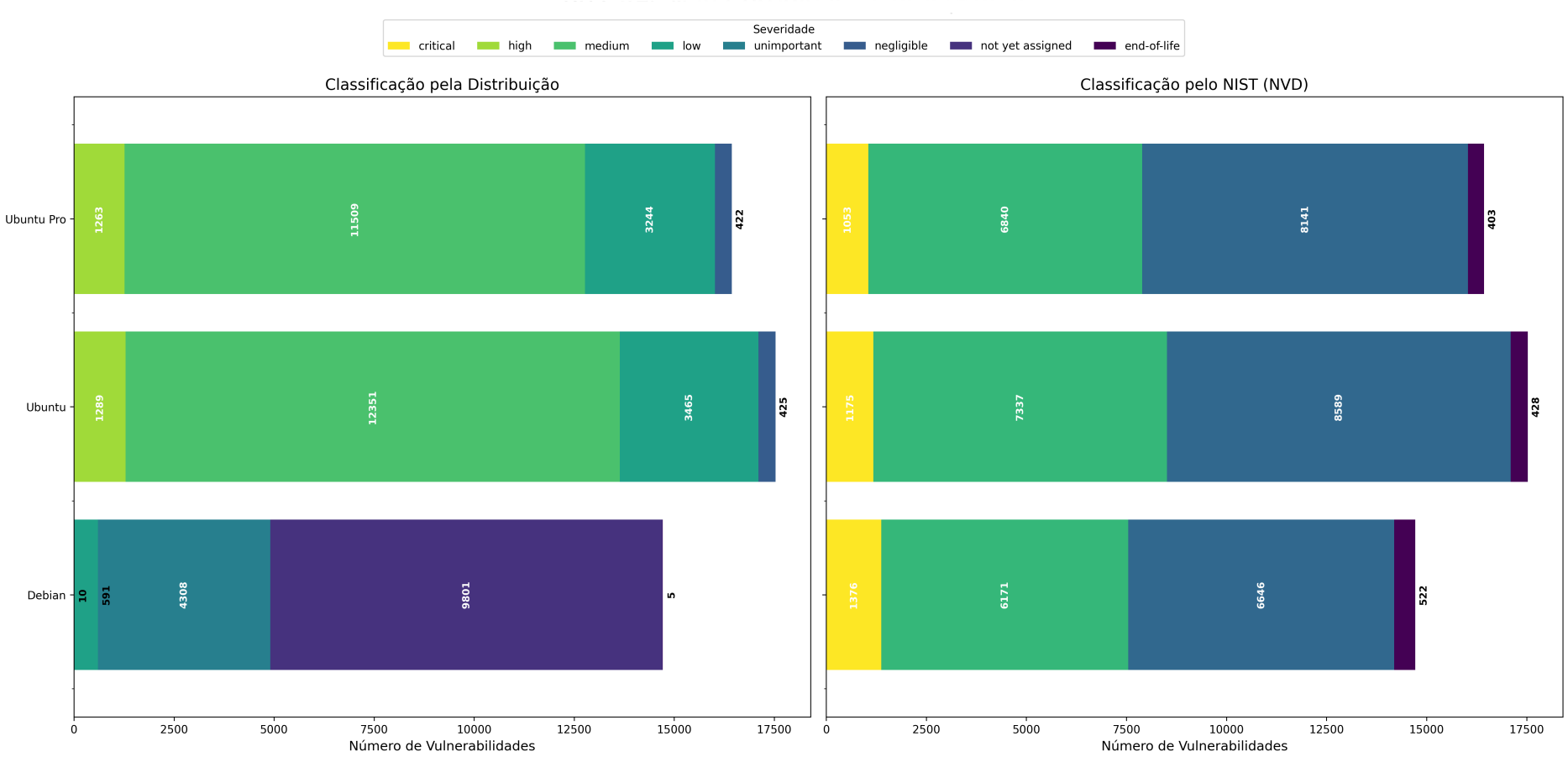
Como já apresentado pela figura 15, essas distribuições não chegam a corrigir integralmente o conjunto de vulnerabilidades presentes no Red Hat. Adicionalmente, não há divulgação das vulnerabilidades pendentes de correção, fato confirmado pela ausência de registros não corrigidos (valor zero) em seus relatórios. Portanto, não é possível considerá-las superiores em termos de eficácia ou transparência na gestão de vulnerabilidades.

Outro aspecto relevante observado tanto no conjunto geral quanto no de vulnerabilidades comuns é o padrão de comportamento similar entre as distribuições, com destaque para o elevado número de vulnerabilidades não corrigidas no Red Hat e a maior incidência de datas negativas no Ubuntu. Esse padrão será analisado em detalhes na seção 5.5.

5.4.1 Criticidade das vulnerabilidades não corrigidas e com datas negativas

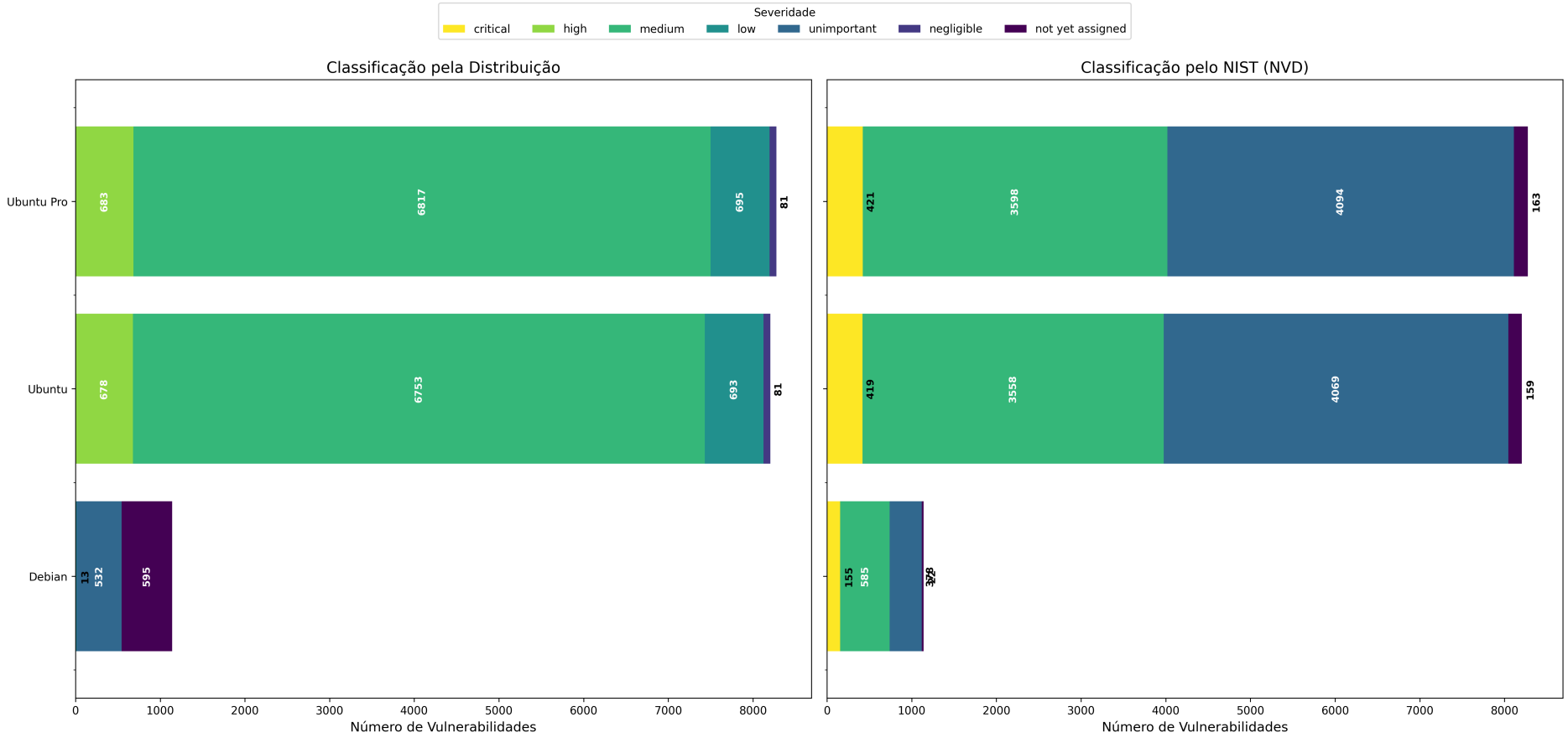
Continuando a análise das vulnerabilidades não corrigidas e com datas negativas, será feita uma análise do cenário geral para entender qual o risco de segurança que essas vulnerabilidades podem representar, conforme apresentado pelas figuras 22, 23, 24 e 25.

Figura 22 – Comparação de Severidades de CVEs Não Corrigidas - Família .deb



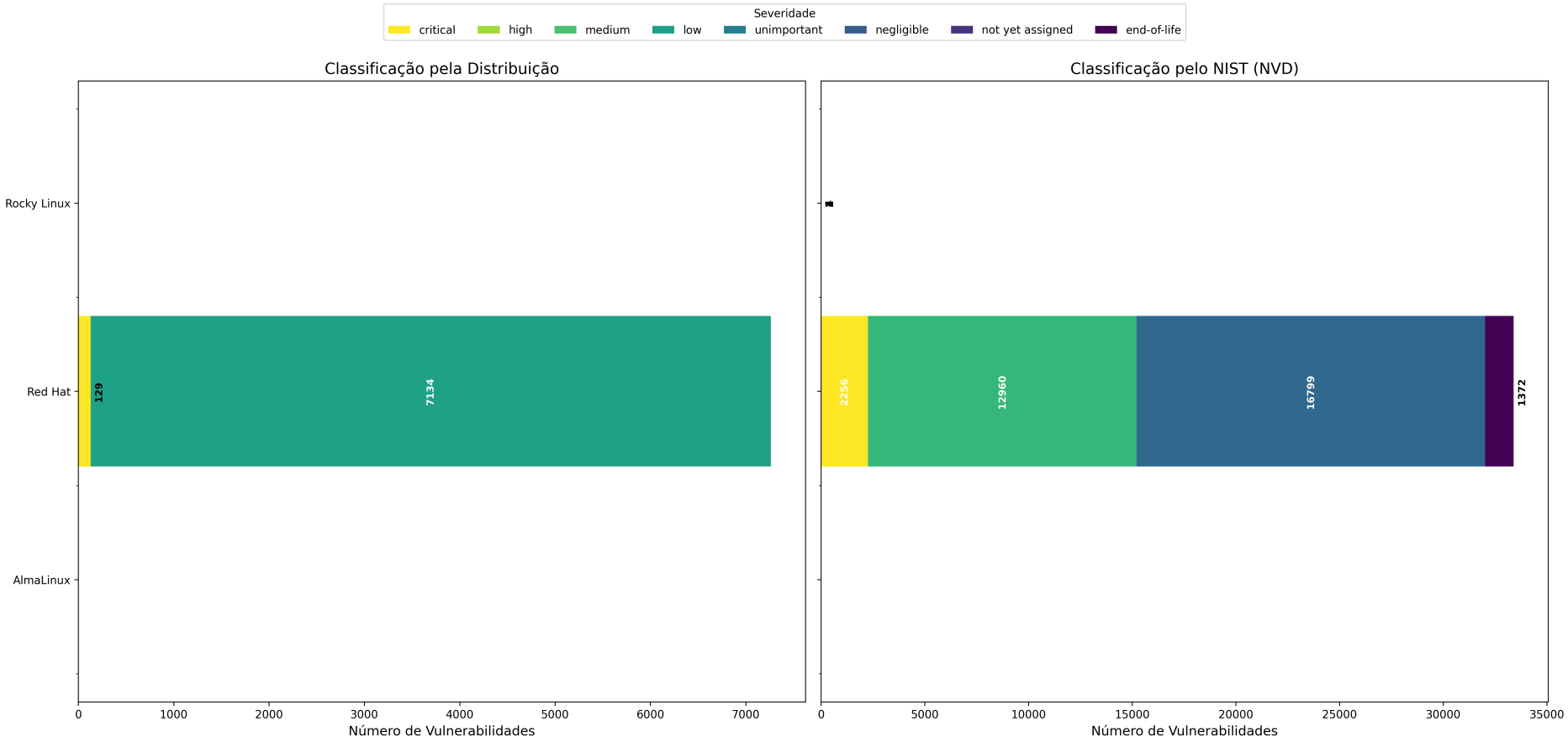
Fonte: Produzido pelo autor

Figura 23 – Comparação de Severidades de CVEs Com Datas Negativas - Família .deb



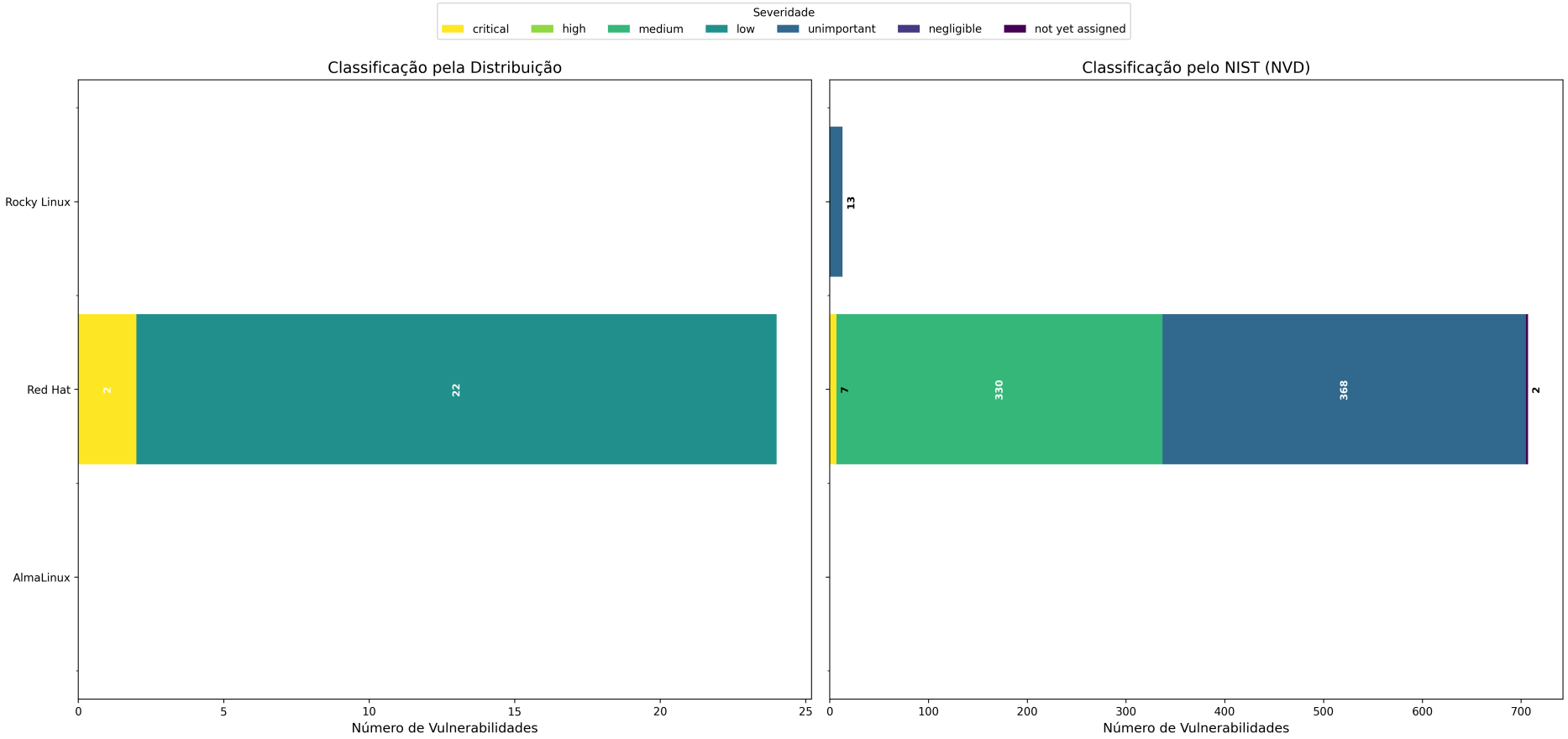
Fonte: Produzido pelo autor

Figura 24 – Comparação de Severidades de CVEs Não Corrigidas - Família .rpm



Fonte: Produzido pelo autor

Figura 25 – Comparação de Severidades de CVEs Com Datas Negativas - Família .rpm



Fonte: Produzido pelo autor

A observação dos dados permite afirmar que o ecossistema DEB apresenta comportamento distinto do Red Hat (considerando que AlmaLinux e Rocky Linux não divulgam vulnerabilidades não corrigidas). Enquanto a maioria das vulnerabilidades não corrigidas no Red Hat possui baixa severidade, e as correções de dia zero contêm uma pequena porcentagem de vulnerabilidades críticas (sendo majoritariamente de baixa severidade), no ambiente DEB percebe-se que as vulnerabilidades não corrigidas e com datas negativas são predominantemente de severidade média. Esta diferença sugere distintas filosofias e capacidades de triagem de segurança entre os ecossistemas.

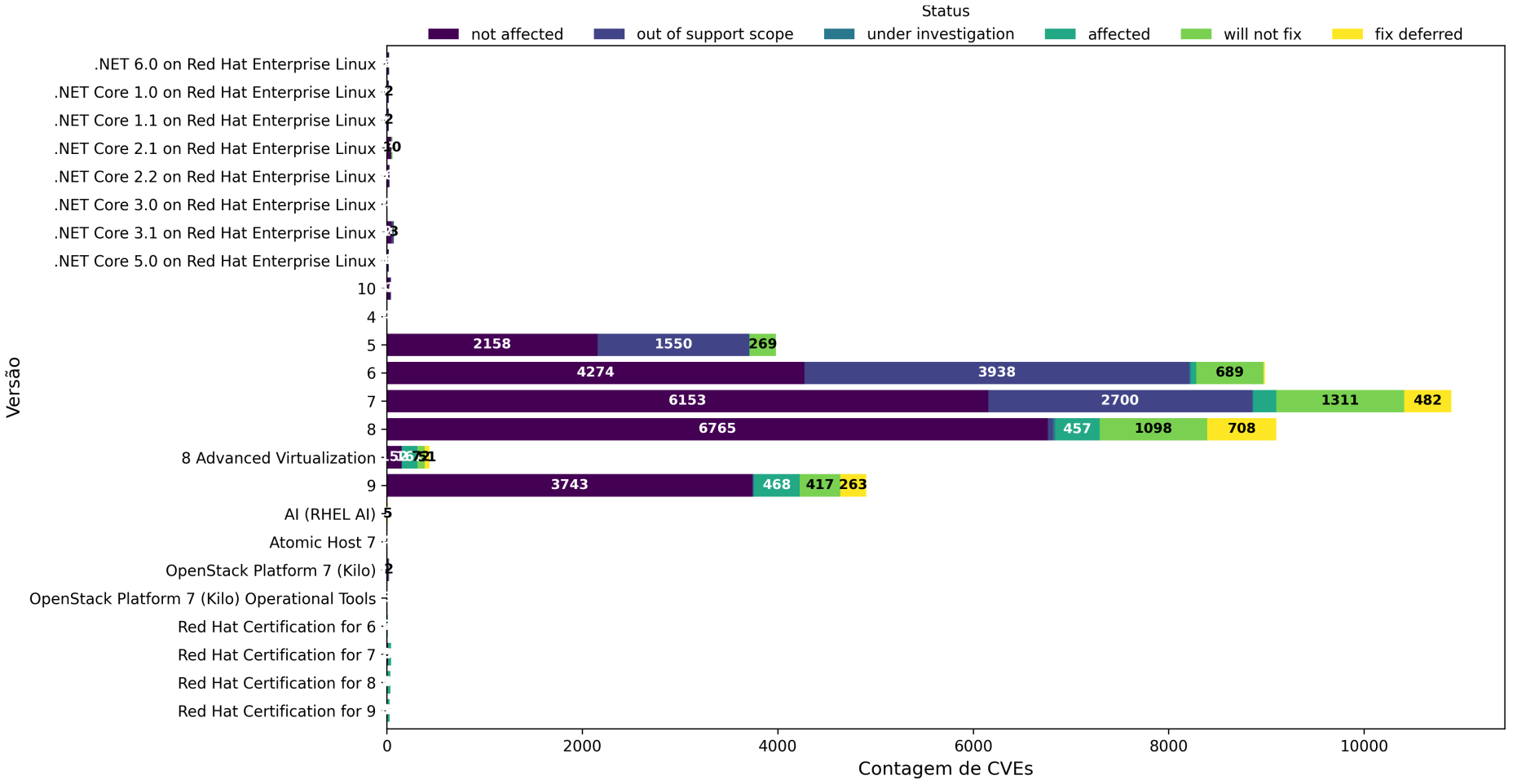
Ressalta-se que o Debian não fornece informações de severidade no momento da coleta de dados, o que dificulta análises mais aprofundadas. Contudo, ao comparar com as classificações do NIST, verifica-se que seu comportamento é semelhante ao do Ubuntu.

Quanto ao Ubuntu e Ubuntu Pro, observa-se a mesma classificação previsível de severidade.

5.5 Análise detalhada do Red Hat e Ubuntu - vulnerabilidades não corrigidas

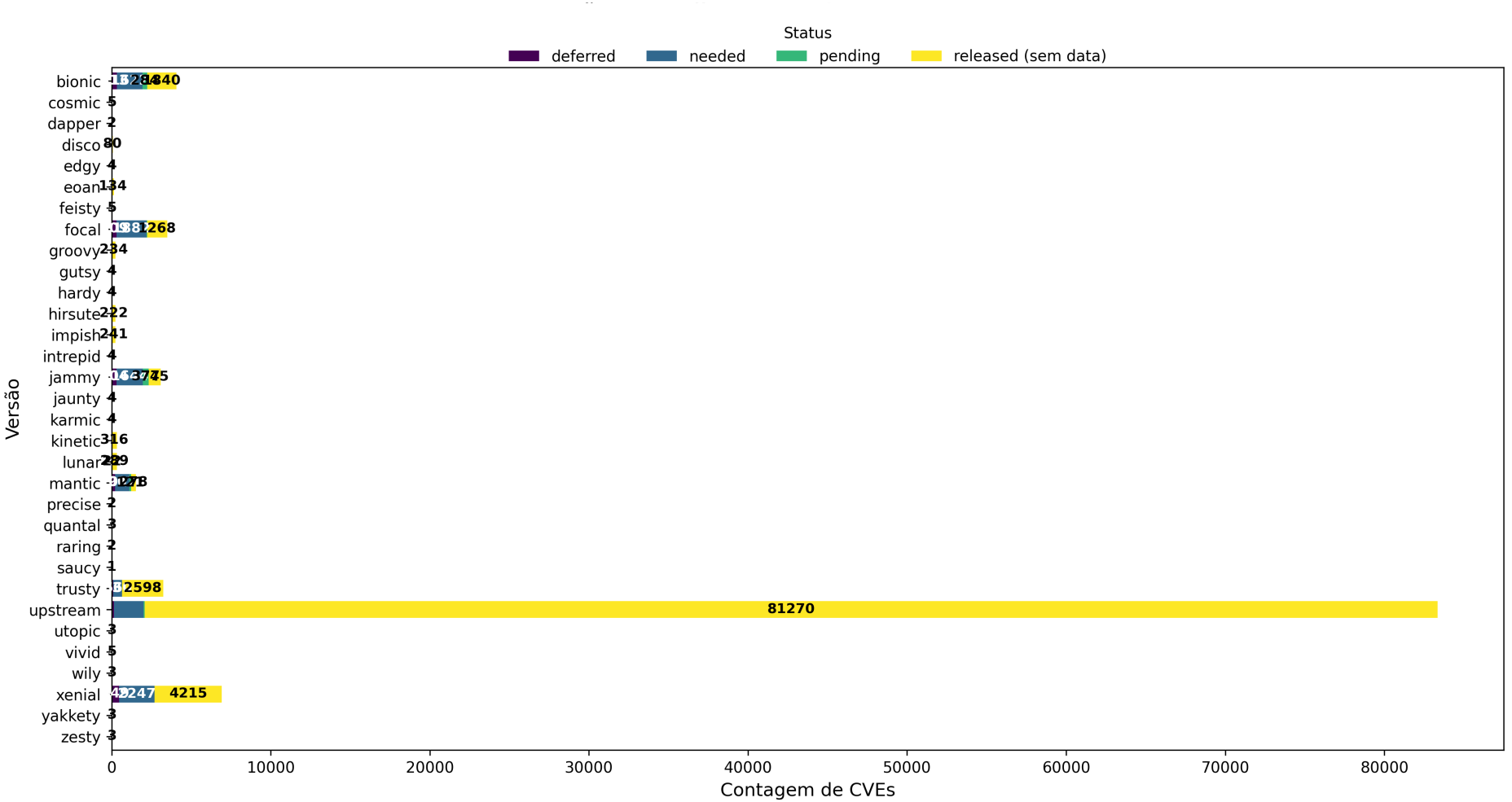
Conforme evidenciado pelas tabelas 6 e 7, o volume significativo de vulnerabilidades não corrigidas identificado no Ubuntu e no Red Hat mostrou-se relevante para uma investigação mais aprofundada. Para analisar as causas subjacentes a esses valores, procedeu-se à extração das versões dessas distribuições. Os resultados dessa análise são apresentados nas figuras 26 e 27 a seguir.

Figura 26 – Detalhamento de CVEs Não Corrigidos - Red Hat



Fonte: Produzido pelo autor

Figura 27 – Detalhamento de CVEs Não Corrigidos - Ubuntu



Fonte: Produzido pelo autor

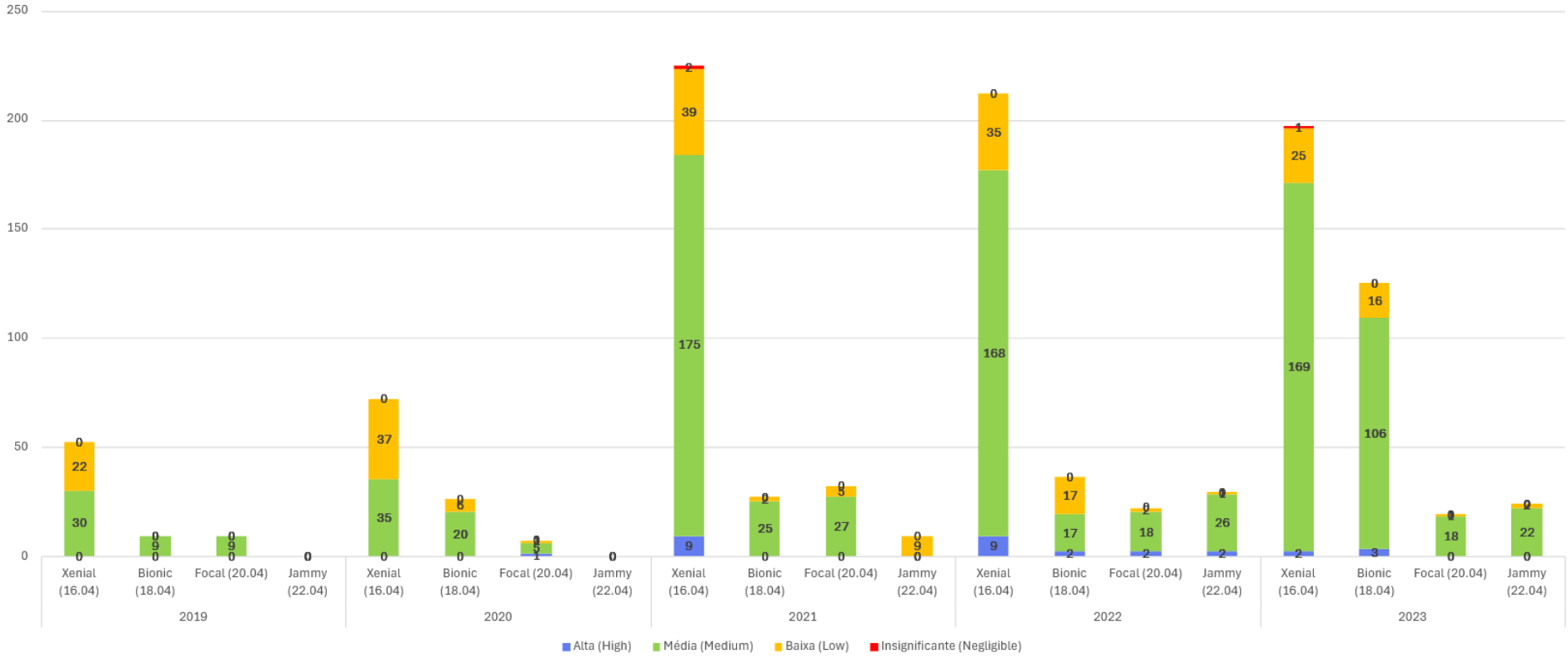
A figura 26 revela um processo de divulgação maduro e transparente por parte da Red Hat. Observa-se que a grande maioria das vulnerabilidades "não corrigidas" encontra-se categorizada como *Not Affected*, indicando que os pacotes em questão não são efetivamente vulneráveis na distribuição, ou como *Out of Support Scope*, referente a pacotes que, embora vulneráveis, atingiram o fim de seu ciclo de vida e não recebem mais suporte. Os demais status apresentam um fluxo de trabalho previsível e alinhado às boas práticas de gestão de segurança. Em contrapartida, a análise do Ubuntu pela figura 27 revela que a maior parte dos casos não resolvidos refere-se à versão *upstream*, as quais registram o estado de resolvido, mas não divulgam a data de resolução. No contexto de desenvolvimento de software livre, *upstream* (54) refere-se aos projetos de software originários dos quais o Ubuntu deriva seus pacotes. Sendo uma distribuição *downstream*, o Ubuntu mantém relações de colaboração fundamentais com seus diversos *upstreams* (notadamente a Debian e projetos como o kernel Linux). O objetivo dessa colaboração é manter os pacotes atualizados, compartilhar melhorias (processo conhecido como *upstreaming*) e beneficiar-se do controle de qualidade realizado pela comunidade mais ampla.

Ressalta-se, contudo, que a ausência de divulgação das datas de correção da versão *upstream* representa uma limitação na transparência das informações, dificultando a avaliação precisa do tempo de resposta às vulnerabilidades.

5.6 Análise da criticidade de vulnerabilidades corrigidas apenas no Ubuntu Pro

Conforme evidenciado pelas tabelas 6 e 7, as vulnerabilidades corrigidas exclusivamente no Ubuntu Pro representam uma diferença quantitativa modesta (entre 1 e 3 pontos percentuais). Contudo, a análise da criticidade dessas vulnerabilidades (os dados foram extraídos e processados por meio do código desenvolvido analyze_pro_only_fixes.py) possibilita avaliar o risco adicional ao qual estão expostos os usuários da versão padrão do Ubuntu sem assinatura Pro. A figura 28 ilustra essa análise de criticidade.

Figura 28 – Corrigidas apenas no Ubuntu Pro - Geral



Fonte: Produzido pelo autor

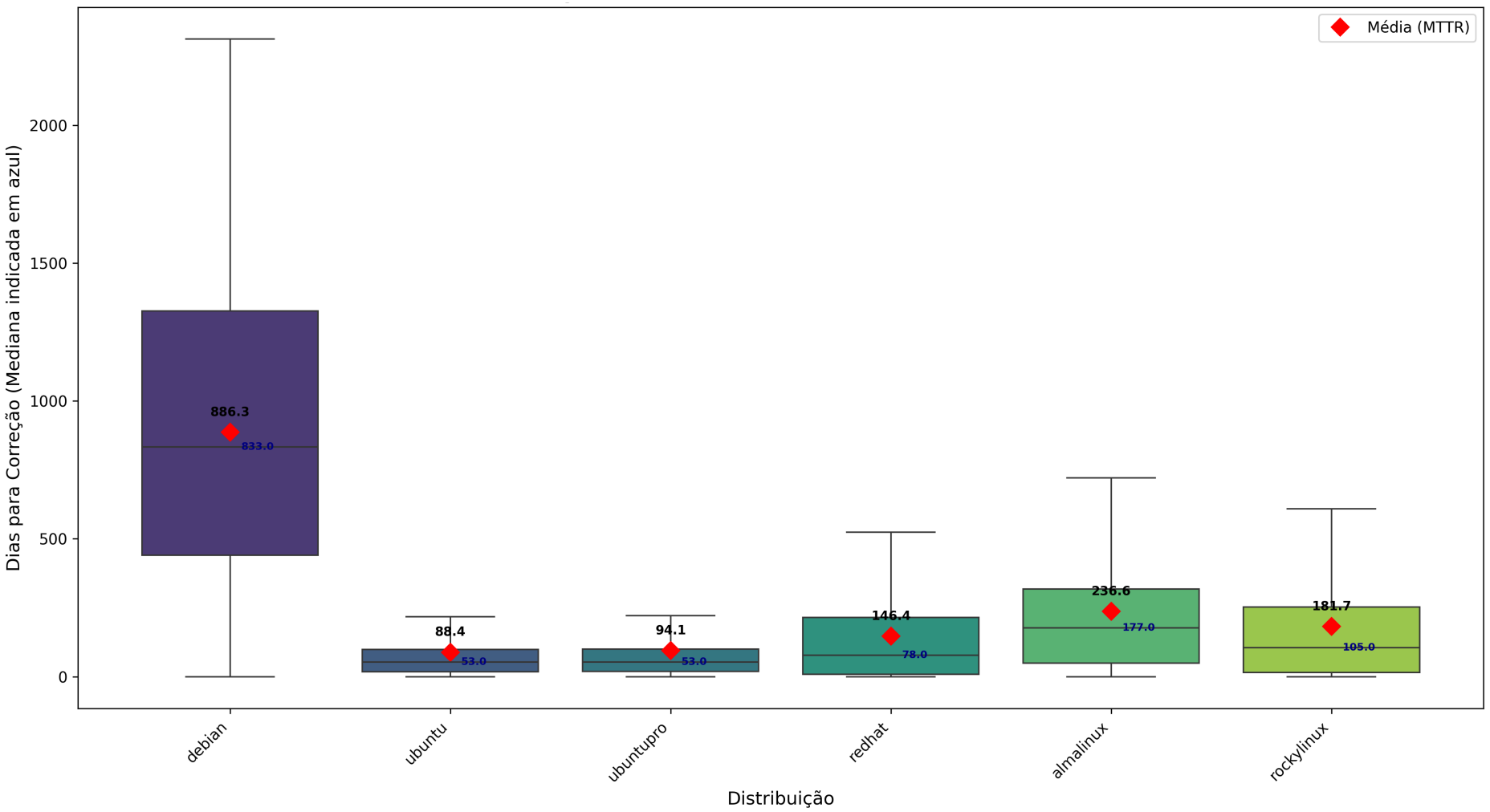
A análise dos gráficos evidencia limitações na cobertura de segurança da versão padrão do Ubuntu. Constata-se que, mesmo dentro do período de suporte, vulnerabilidades de severidade média (a maioria) não são sanadas, o que confirma a política da Canonical de reservar tais correções para os assinantes do Ubuntu Pro (49).

Esse cenário se agrava com o fim do ciclo de atualizações padrão. A partir desse momento, o suporte estendido é uma exclusividade dos clientes Pro, garantindo-lhes uma cobertura de segurança abrangente durante todo o ciclo de vida da versão, enquanto os usuários da versão comunidade ficam desprotegidos.

5.7 Apresentação do tempo médio de correção

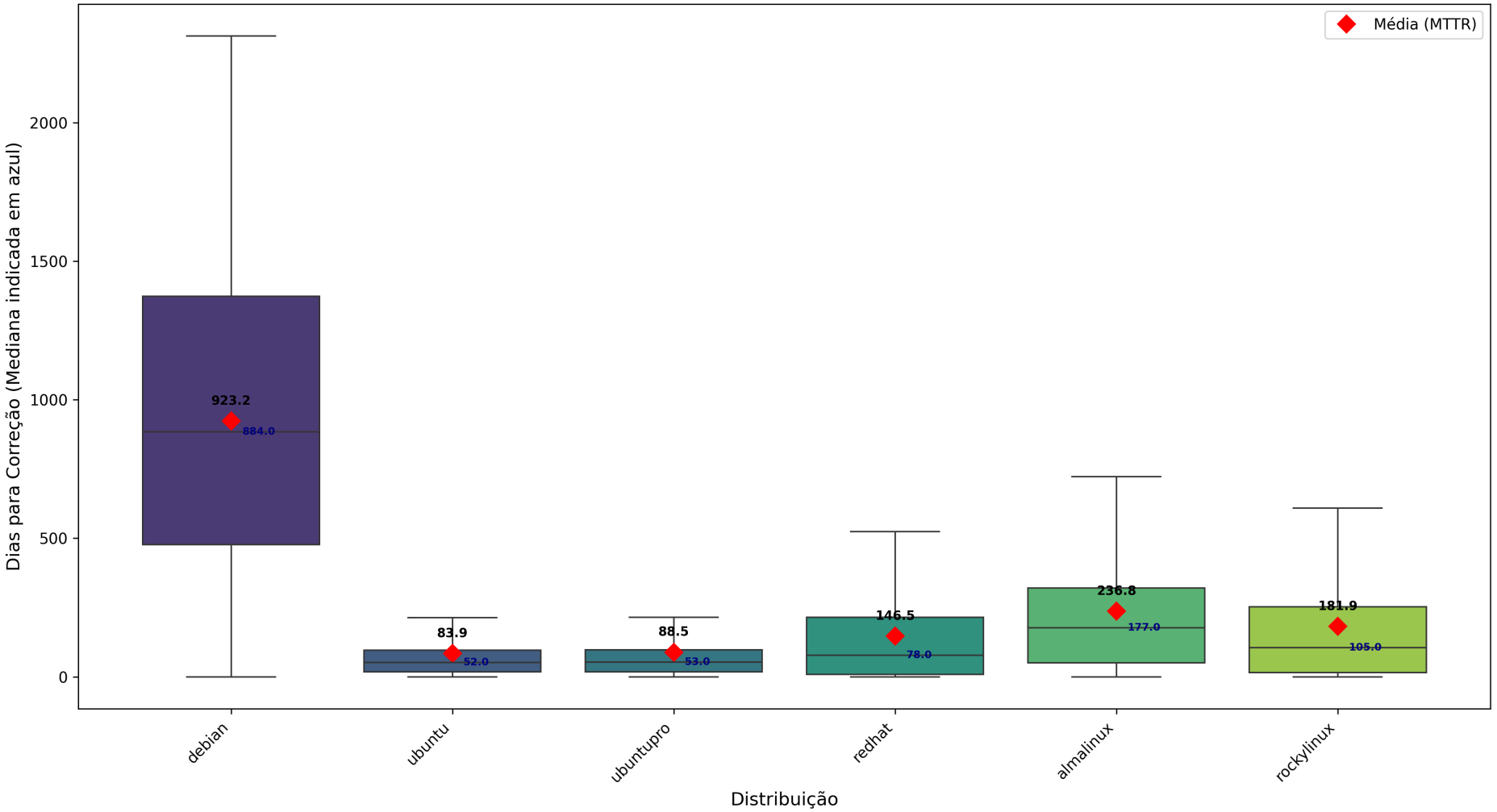
Após as análises aprofundadas das vulnerabilidades e suas características, que permitiram compreender o comportamento de cada distribuição Linux por meio do método ColATeC, apresentam-se nesta seção os tempos médios de correção calculados, concluindo assim as questões de pesquisa remanescentes. As figuras 29 e 30 apresentam o desempenho geral no período estudado, calculando as médias gerais do tempo de correção das vulnerabilidades gerais e de pacotes em comum.

Figura 29 – Tempo de Correção Geral - 5 Anos



Fonte: Produzido pelo autor

Figura 30 – Tempo de Correção Pacotes em Comum - 5 Anos



Fonte: Produzido pelo autor

A análise dos tempos de resolução, apresentada nas figuras anteriores, revela três conclusões principais:

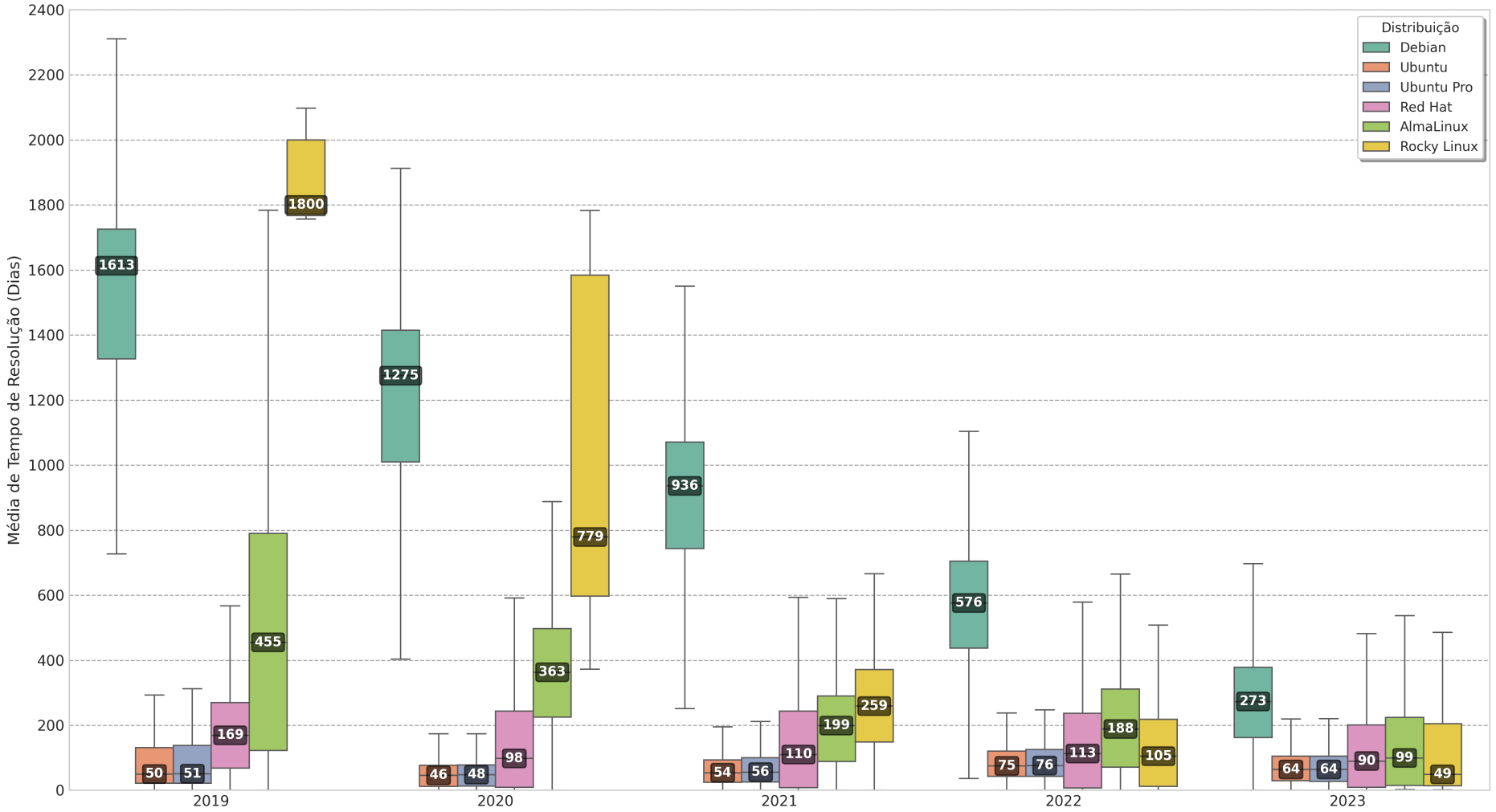
i) Há uma notável consistência no perfil de desempenho de cada distribuição entre o cenário geral e o de pacotes comuns, reforçando que as observações refletem características intrínsecas de cada ecossistema. Isto corrobora a validade da análise geral, indicando que o desempenho observado não constitui um artefato de pacotes específicos, mas sim uma característica inerente aos processos de gestão de segurança de cada distribuição;

ii) A distribuição Debian destaca-se como um caso atípico, apresentando não apenas uma mediana de tempo de correção superior, mas também uma dispersão temporal significativamente maior. Enquanto algumas vulnerabilidades são corrigidas rapidamente, uma parcela considerável demanda prazos substancialmente mais longos para resolução;

iii) O ecossistema RPM apresenta forte coesão interna, com a Red Hat a liderar em desempenho e as suas derivadas, AlmaLinux e Rocky Linux, apresentando resultados praticamente equivalentes e muito próximos aos da distribuição matriz.

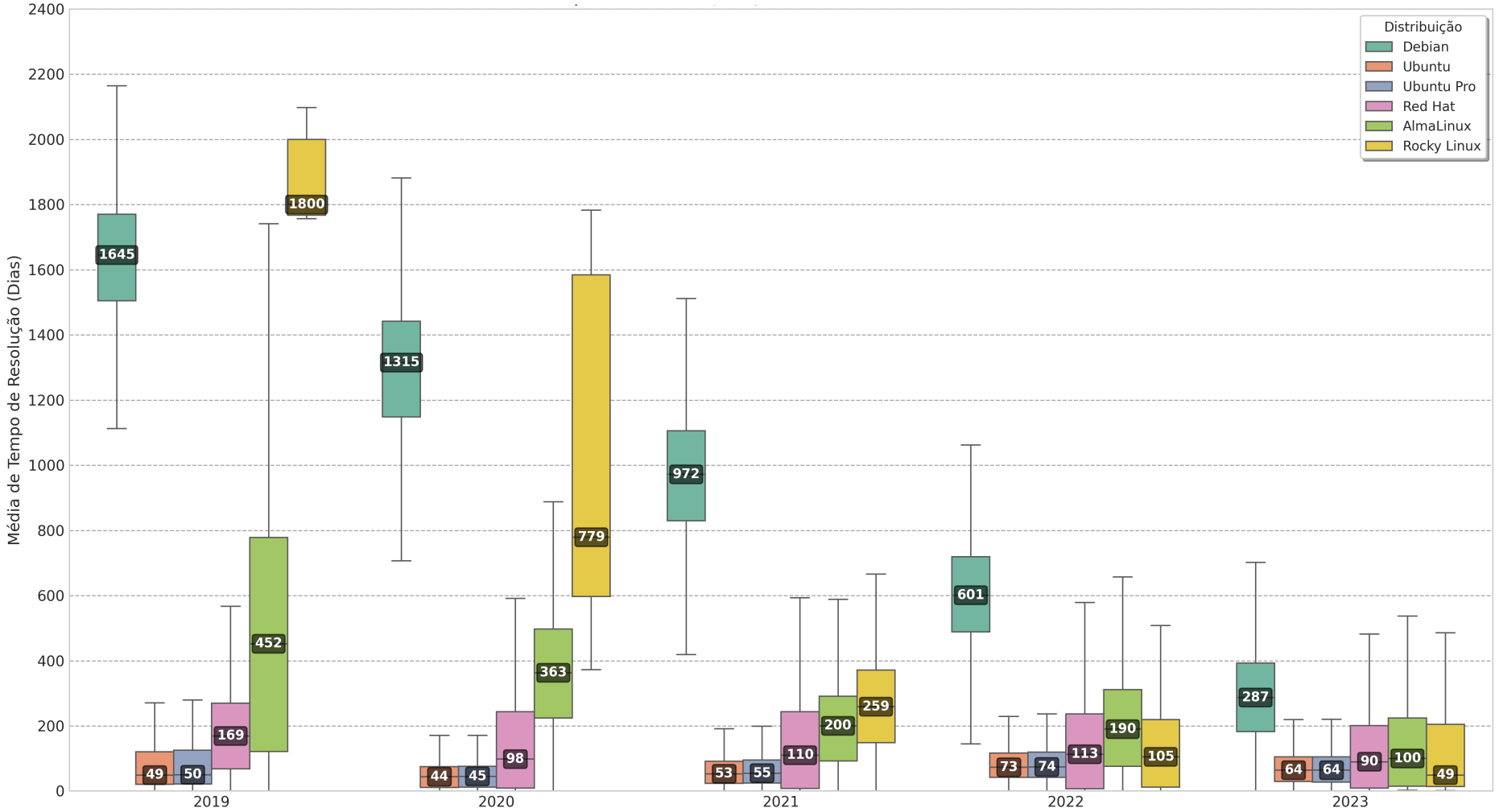
Contudo, ainda não é possível determinar qual distribuição corrige suas vulnerabilidades com maior rapidez. Considerando que o método ColATeC possibilita a coleta sistemática de dados de vulnerabilidades, torna-se viável realizar análises comparativas adicionais. Para uma investigação temporal mais detalhada, as figuras 31 e 32 apresentam o desempenho ano a ano do tempo de correção, utilizando o campo ***Year*** derivado diretamente do atributo *Published* do NVD como referência temporal.

Figura 31 – Tempo de Correção Geral - Ano a Ano



Fonte: Produzido pelo autor

Figura 32 – Tempo de Correção Pacotes em Comum - Ano a Ano



Fonte: Produzido pelo autor

A análise das figuras 31 e 32 permite concluir o seguinte:

1. Ubuntu e Ubuntu Pro foram as distribuições que corrigiram mais rapidamente suas vulnerabilidades durante o período estudado;

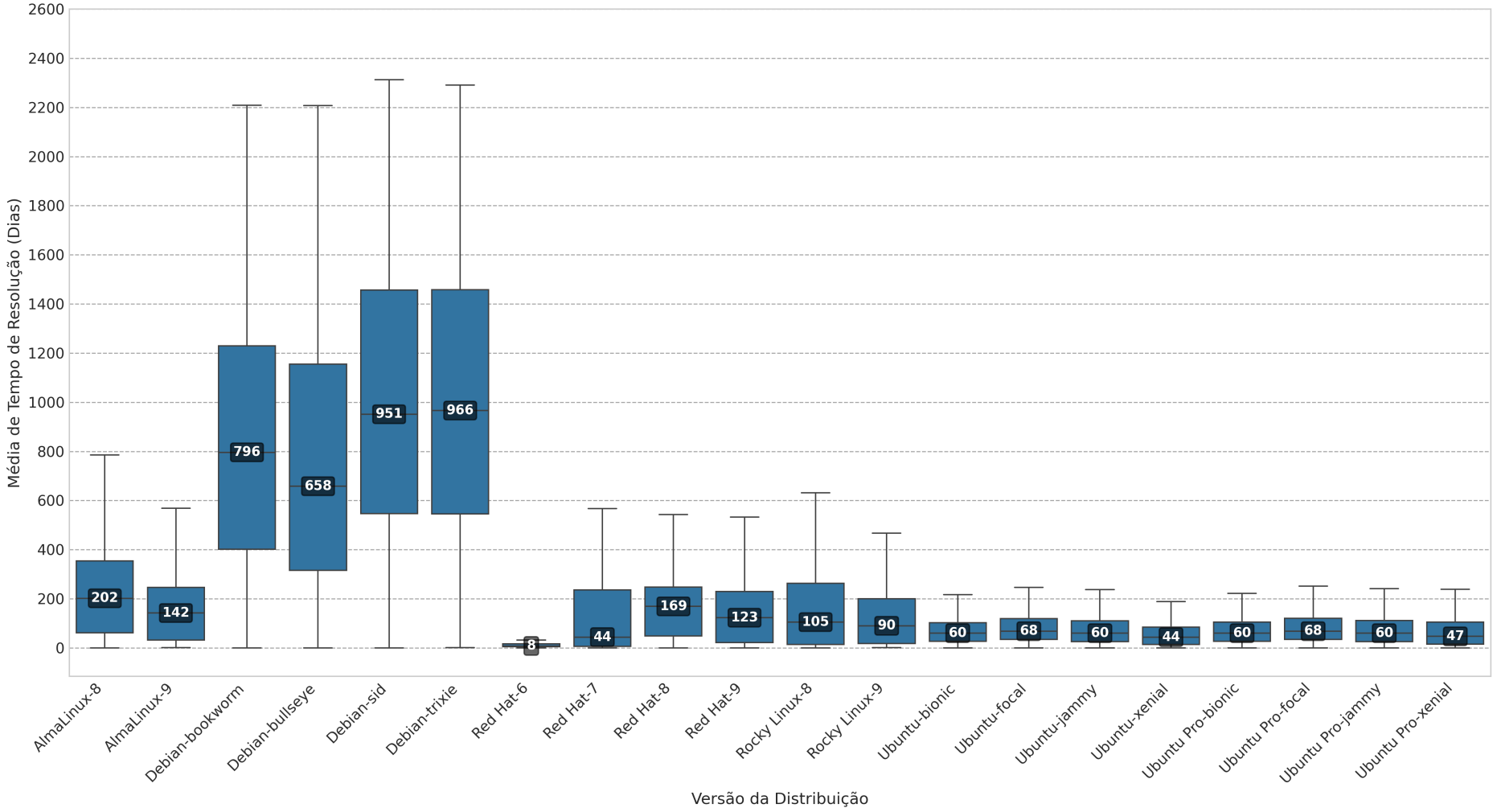
2. A Red Hat apresentou tempos de correção estáveis, com baixa variabilidade e valores comparáveis aos do Ubuntu;

3. AlmaLinux e Rocky Linux iniciaram com tempos elevados, condizentes com seu período inicial de desenvolvimento (conforme apresentado nas seções 5.2 e 5.2.1), mas em 2022 e 2023 atingiram o padrão da Red Hat. Destaca-se o Rocky Linux, que chegou a corrigir mais rapidamente, embora, conforme discutido na seção 5.3, ambas tenham registrado um número significativamente menor de vulnerabilidades;

4. O Debian vem melhorando progressivamente seu tempo de correção ao longo do período analisado, indicando uma possível otimização em seus processos de correção de vulnerabilidades.

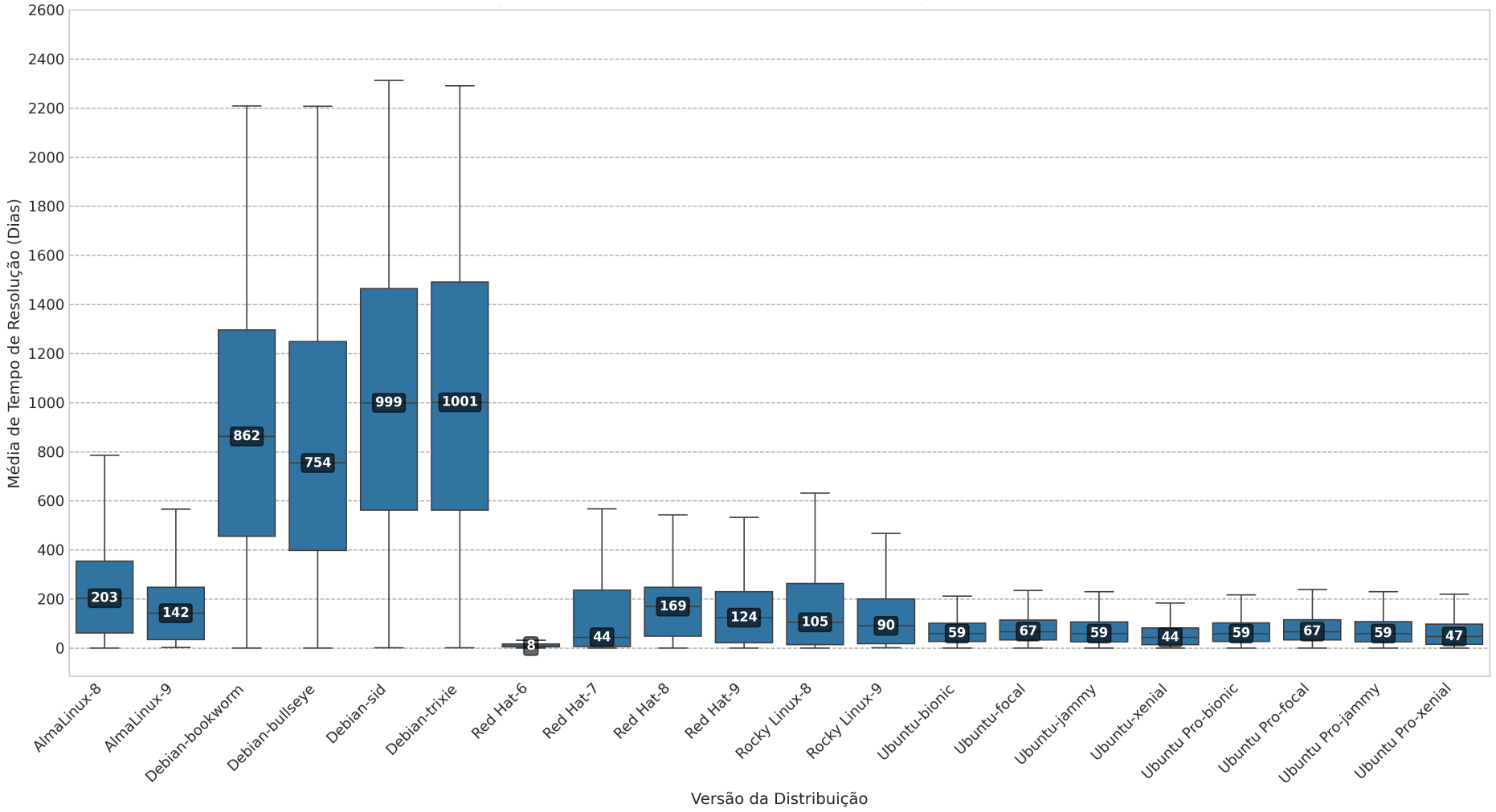
A seguir, as figuras 33, 34, 35 e 36 (geradas pelo código-fonte results3.py) apresentam uma análise temporal detalhada baseada nas versões de cada distribuição, com o objetivo de compreender e concluir as análises do estudo de caso.

Figura 33 – Comparação Agregada - Geral



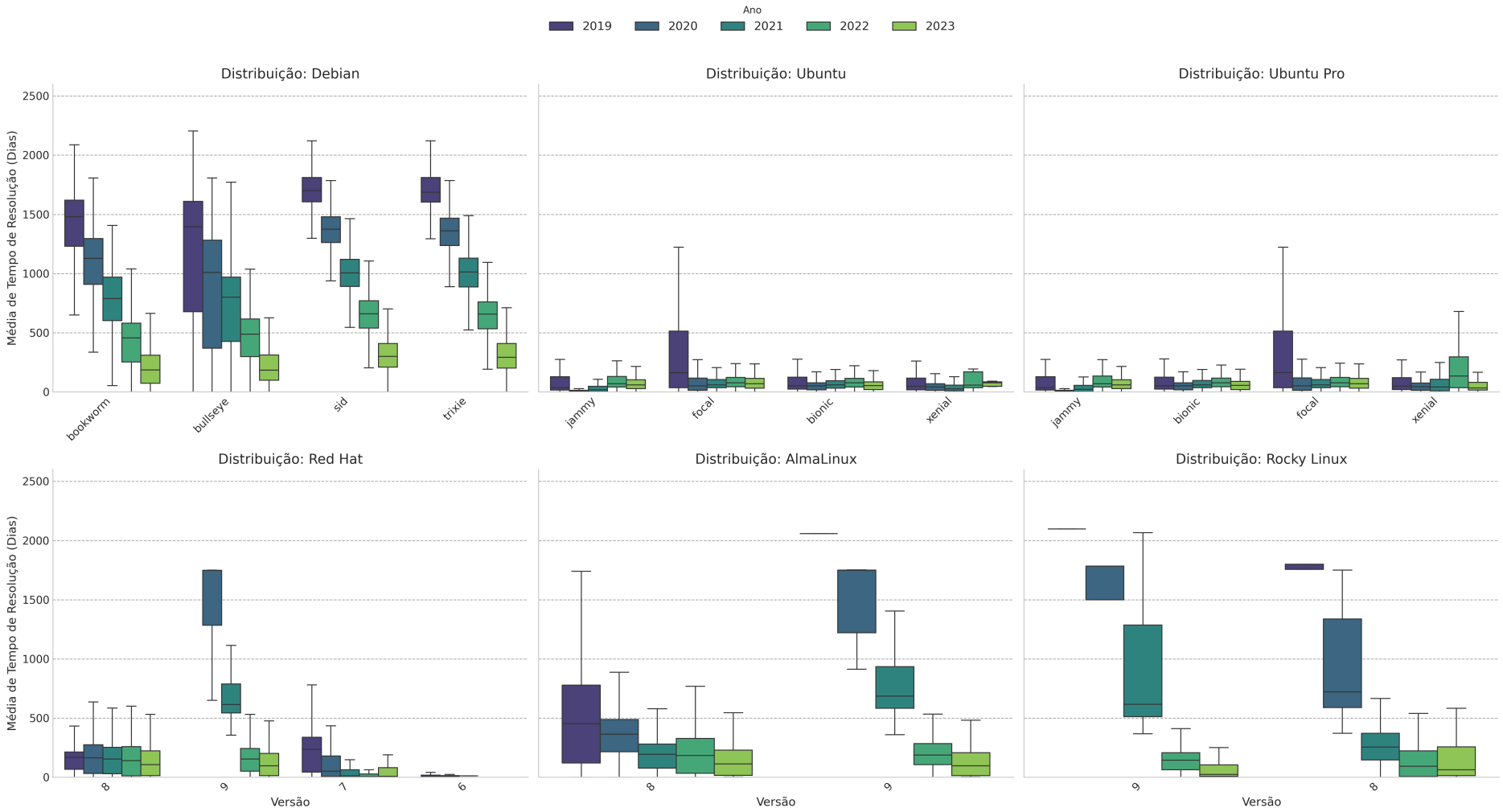
Fonte: Produzido pelo autor

Figura 34 – Comparação Agregada - Pacotes em Comum



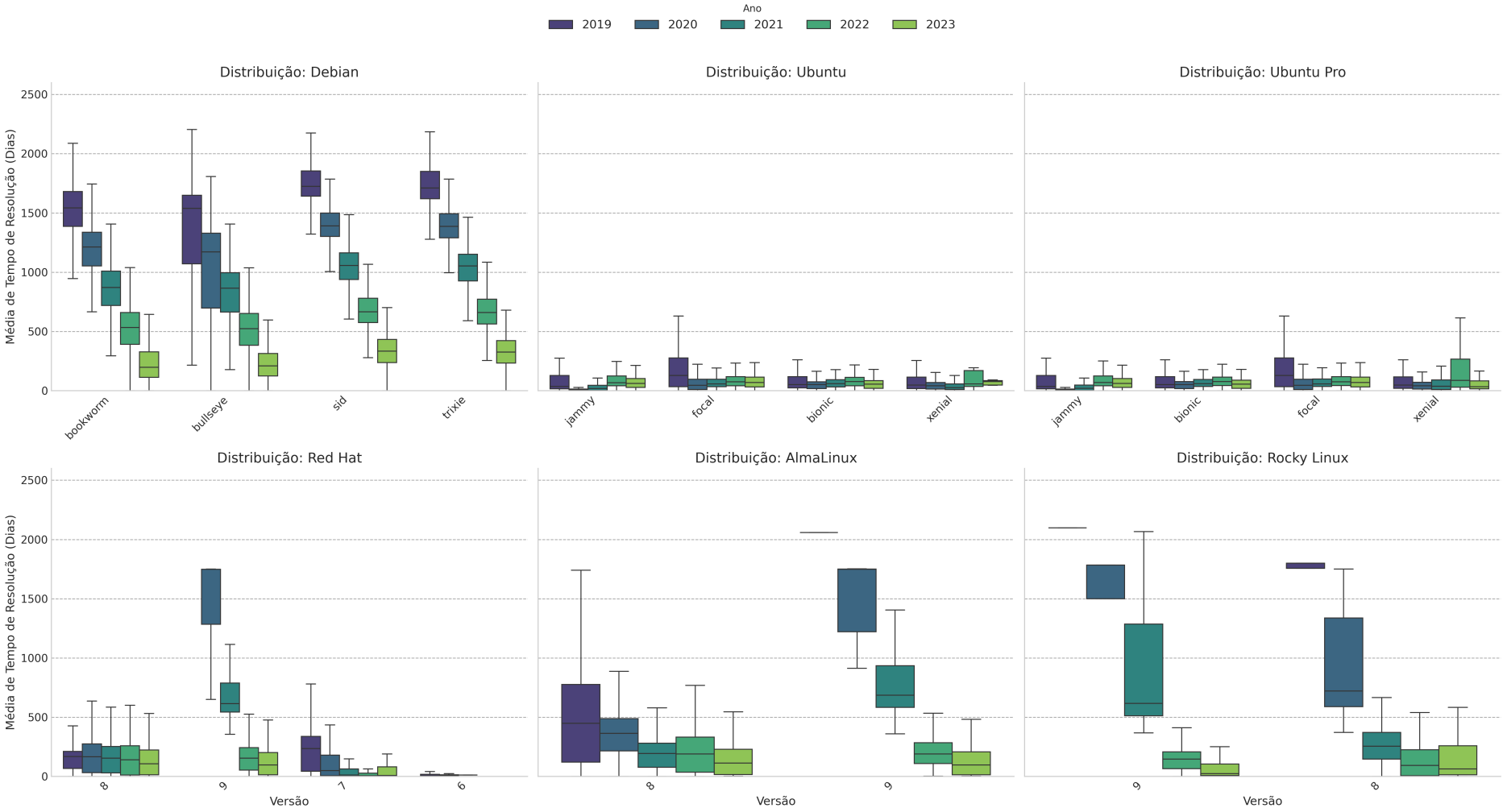
Fonte: Produzido pelo autor

Figura 35 – Comparação Agregada Versões - Geral



Fonte: Produzido pelo autor

Figura 36 – Comparação Agregada Versões - Pacotes em Comum



Fonte: Produzido pelo autor

A análise das figuras 33, 34, 35 e 36 indica o comportamento das versões das distribuições e permite concluir o seguinte:

i) Os tempos de correção variam entre as diversas versões de uma mesma distribuição Linux, fato explicado pelos seguintes aspectos:

   a) No caso da Red Hat, a versão 9 (e suas similares AlmaLinux e Rocky Linux) apresentou tempos elevados inicialmente (ano de 2020), embora seu lançamento oficial tenha ocorrido apenas em 2022 (55). Isto indica que, durante o período de desenvolvimento dessa versão, os tempos de correção são mais elevados, não sendo recomendada sua utilização em ambientes de produção durante esta fase;

   b) Comportamento similar foi observado no Ubuntu, onde a versão Focal (20.04 LTS) (56), lançada em 2020, apresentou o mesmo padrão da Red Hat 9 durante sua fase de desenvolvimento em 2019;

   c) O Ubuntu Xenial (16.04 LTS) apresentou valores acima do esperado em 2022 na versão Pro, sendo relevante considerar que esta versão encontrava-se no primeiro ano de suporte estendido (56), o que pode explicar esse comportamento atípico;

   d) Algumas versões antigas, como Red Hat 6 e Ubuntu Xenial (16.04 LTS), apresentam supostamente bons tempos de resolução. No entanto, é necessário compreender que esses valores estão relacionados ao término de seus ciclos de suporte, indicando que, em vez de refletirem eficiência, representam a ausência de correções de segurança completas, tornando-as vulneráveis a falhas de segurança.

Ao final deste capítulo, é possível responder às questões de pesquisa remanescentes, uma vez que a aplicação do método ColATeC permitiu realizar análises aprofundadas de cada distribuição Linux estudada, chegando-se às seguintes conclusões:

– Q4: "*Qual distribuição possui o menor tempo médio de correção de vulnerabilidades?*": O Ubuntu apresentou o melhor desempenho geral, mantendo-se como a distribuição mais rápida na correção de vulnerabilidades, mesmo considerando os registros não analisados (conforme tabela 5). Ressalta-se que o Ubuntu Pro apresenta uma vantagem adicional, corrigindo um número maior de vulnerabilidades em comparação com a versão sem assinatura (57);

– Q5: "*Existe tratamento diferenciado para versões distintas de uma mesma distribuição?*": Verifica-se que sim, as distribuições podem apresentar desempenhos distintos na correção de vulnerabilidades entre diferentes versões, geralmente melhorando com o tempo. Recomenda-se evitar o uso de versões em fase de desenvolvimento ou com

ciclo de vida de suporte encerrado, uma vez que essas condições implicam maior exposição a vulnerabilidades de segurança.

Capítulo 6

Conclusão

O tempo de correção de vulnerabilidades é um fator crítico para a segurança computacional, independentemente do sistema operacional ou *software* analisado. Entretanto, a ausência de metodologias padronizadas para coleta e análise comparativa dessas métricas entre sistemas distintos motivou o desenvolvimento do método ColATeC, que estabelece padrões mínimos para a disponibilização, coleta e processamento de dados de vulnerabilidades, aplicáveis a quaisquer *softwares* (não apenas a distribuições Linux). Ao introduzir um processo automatizado e auditável para normalização, coleta e validação de métricas temporais, o método apresentou viabilidade e replicabilidade em ambientes heterogêneos, conforme evidenciado pelo estudo de caso comparativo das distribuições Linux Red Hat, AlmaLinux, Rocky Linux, Debian e Ubuntu.

Os resultados apresentados, que quantificam o tempo de correção de vulnerabilidades em distribuições Linux, devem ser interpretados dentro de um contexto econômico e de incentivos mais amplos. Conforme argumentado por RESCORLA (8), a morosidade na aplicação de *patches* não é apenas uma falha técnica, mas uma consequência racional de um sistema em que os usuários arcam com os custos da atualização e os fornecedores não são suficientemente responsabilizados pela segurança de seus produtos. Nesse cenário, ferramentas como o método ColATeC tornam-se essenciais, pois, ao trazer transparência e permitir a comparação objetiva do desempenho de diferentes fornecedores (neste caso, as distribuições), funcionam como um mecanismo de transparência. Ao quantificar a janela de vulnerabilidade, este trabalho contribui para o alinhamento de incentivos, estimulando fornecedores a otimizar seus processos e fornecendo aos usuários subsídios para decisões mais embasadas.

Ao fornecer medições precisas e objetivas do tempo de correção de vulnerabilidades, o método ColATeC não apenas serve como uma ferramenta de contabilização e comparação,

mas também gera dados de entrada essenciais para modelos de gestão de risco mais sofisticados. Os dados sobre o tempo para correção de diferentes distribuições podem alimentar diretamente os modelos de simulação propostos por BERES, Y. et al. (18), permitindo que organizações que utilizam Linux em larga escala modelem o impacto de suas escolhas de sistema operacional e otimizem suas políticas de segurança interna. Assim, este trabalho contribui com uma peça fundamental para a tomada de decisões baseada em dados no gerenciamento de vulnerabilidades.

A análise apresentou que existem variações significativas na agilidade com que as distribuições Linux gerenciam e corrigem suas falhas, impactando diretamente o tempo de exposição ao risco para seus usuários. Esta observação está em conformidade com as conclusões de trabalhos seminais na área, como ARBAUGH, W. et al. (5) que já no início dos anos 2000 apontavam que a implantação tardia de correções, e não a falta delas, era a principal causa para a persistência de sistemas vulneráveis. Assim, o método ColATeC não apenas oferece uma ferramenta para avaliação contemporânea, mas também quantifica um desafio crônico na segurança cibernética. Esses resultados permitem que usuários finais escolham sistemas operacionais com base em evidências quantitativas, priorizando segurança e eficiência. Além de orientar usuários, o ColATeC incentiva melhores práticas de desenvolvimento ao:

i) **Expor diferenças de desempenho:** Comparações entre concorrentes estimulam equipes a otimizar processos de correção;

ii) **Detectar inconsistências em dados públicos:** Como no caso da versão Debian Buster, cuja performance anômala foi identificada após análise metodológica;

iii) **Revelar falhas na divulgação de datas de correção:** conforme apresentado nas tabelas 4 e 5, a ausência de dados estruturados (e.g., JSON) impede a análise completa das vulnerabilidades.

Os resultados alcançados reforçam três contribuições centrais:

i) **Padronização de Dados:** Definição de campos obrigatórios (ex.: CVE, datas de publicação e resolução) e a adoção do formato JSON criaram um fluxo interoperável, facilitando a integração de dados de fontes heterogêneas;

ii) **Escalabilidade:** A automatização via códigos em Python comprovou a capacidade de coletar e validar grandes volumes de dados sem intervenção manual contínua;

iii) **Transparência Reproduzível:** Rastreabilidade técnica e temporal (via UTC+0) garantiu a possibilidade de análises independentes, fortalecendo a confiabilidade dos resultados.

Apesar dos avanços, o método ainda depende de fontes estruturadas e completas: repositórios com metadados ausentes exigem validação manual, o que amplia o tempo de processamento; as distribuições cobertas concentram-se no período de suporte ativo entre 2019 e 2023, não retratando ciclos de manutenção anteriores; e a métrica de tempo de correção não pondera a criticidade individual das vulnerabilidades. Essas limitações orientam leituras cuidadosas dos resultados e priorizam evoluções futuras.

Para facilitar replicações, o repositório público disponibiliza tanto o código-fonte quanto o *dump* da base processada, permitindo auditoria independente das etapas de coleta e transformação sem repetir o esforço de extração original.

Recomenda-se monitorar, como indicador de impacto, o número de distribuições que aderem integralmente ao conjunto mínimo de campos definidos pelo ColATeC e a variação anual do tempo mediano de correção após a adoção da metodologia, métricas que aproximam os ganhos observados da redução efetiva de risco.

Do ponto de vista acadêmico, este trabalho oferece um *framework* metodológico aberto e documentado, passível de extensão para outras plataformas (ex.: Internet of Things (IoT), BSD). Na prática, a padronização proposta beneficia equipes de segurança ao reduzir o tempo gasto na normalização manual de dados, permitindo foco em análises de risco e mitigação proativa.

Embora o ColATeC tenha sido validado em cenários reais, sua adoção em larga escala depende do engajamento da comunidade de segurança e de mantenedores de sistemas.

Por fim, este trabalho reforça a premissa de que a transparência e a automação são pilares essenciais para enfrentar a complexidade crescente das ameaças cibernéticas. O ColATeC não apenas responde a uma demanda imediata da área, mas também abre caminho para pesquisas interdisciplinares que unifiquem segurança, ciência de dados e políticas de governança.

6.1 Trabalhos Futuros

O método ColATeC estabelece um padrão promissor para coleta e análise de vulnerabilidades, mas abre caminho para trabalhos complementares. As seguintes direções são propostas para ampliação e refinamento do método:

1. **Expansão de Escopo**

   - **Inclusão de Novos Sistemas:** Aplicar o ColATeC a outras famílias de sistemas operacionais e coleções de *software* (ex.: Ports dos diversos BSDs, MacPorts e Homebrew no macOS, Chocolatey no Windows, entre outros), avaliando a adaptabilidade do método em ecossistemas heterogêneos;

- **Integração com Dispositivos IoT:** Investigar a viabilidade de coletar dados de vulnerabilidades em dispositivos IoT, que frequentemente possuem ciclos de atualização distintos.

2. **Aprimoramento da Automatização**

   - **Validação com Inteligência Artificial**: Utilização de modelos de aprendizado de máquina e *Large Language Models (LLM)* para mineração de dados de vulnerabilidades em fontes não estruturadas, incluindo listas de discussão, ferramentas de *bug tracker* e sites web;
   - **APIs Especializadas:** Desenvolver APIs padrão para integração direta com ferramentas de segurança (ex.: *Security Information and Event Management (SIEM)*s, *scanners* como Nessus), acelerando respostas a ameaças;
   - **Metadados de Proveniência:** Adicionar campos para rastrear a origem dos dados (ex.: ferramenta de coleta, responsável pela publicação), aumentando a transparência.

3. **Estudos Longitudinais**

   - **Análise de Impacto Temporal:** Investigar correlações entre a velocidade de correção de vulnerabilidades (via campo data de resolução) e a exploração efetiva em ataques reais;
   - ***Benchmarking* de Distribuições:** Comparar o desempenho de distribuições Linux ou de outros *softwares* na gestão de vulnerabilidades usando métricas padronizadas pelo ColATeC.

4. **Promoção e Padronização Comunitária**

   - **Governança Colaborativa:** Criar um conselho comunitário com representantes de distribuições, desenvolvedores de ferramentas e usuários corporativos para validar alterações no esquema de dados e resolver divergências;
   - **Campanhas de Capacitação:** Organizar oficinas periódicas e *webinars* para demonstrar a execução dos coletores, incentivar contribuições e divulgar boas práticas na publicação de dados estruturados;
   - **Parcerias Institucionais:** Firmar acordos com entidades como Linux Foundation e Forum of Incident Response and Security Teams (FIRST) para reconhecer o ColATeC como referência e incluí-lo em guias de conformidade.

5. **Validação em Cenários Reais**

- **Casos de Uso em Ambientes Críticos**: Validar o método em infraestruturas de missão crítica (ex.: sistemas médicos, redes elétricas, aplicações militares) para verificar sua aplicabilidade em contextos com requisitos de segurança distintos dos ambientes convencionais;
- **Simulações de Ataques:** Utilizar dados coletados pelo ColATeC para treinar modelos de risco baseados em cenários realistas de exploração de vulnerabilidades.

Referências

1 SCHWAB, K. **A quarta revolução industrial**. 1ª. ed. São Paulo: Edipro, 2016.

2 CERT.BR. **Estatísticas do CERT.br**. 2025. Disponível em: <https://stats.cert.br>. Acesso em: 01/01/2025.

3 CERT.BR. **Sobre o CERT.br**. 2025. Disponível em: <https://www.cert.br/sobre>. Acesso em: 01/01/2025.

4 NIST. **Glossário - vulnerabilidade**. 2025. <https://csrc.nist.gov/glossary/term/vulnerability>. [Online: acessado em 08/10/2025].

5 ARBAUGH, W. et al. Windows of vulnerability: a case study analysis. **Computer**, v. 33, n. 12, p. 52–59, 2000.

6 CIPHER. **Por que avaliar e corrigir vulnerabilidades conhecidas deve ser prioridade na segurança de dados críticos?** 2025. Disponível em: <https://www.cipher.com/pt_BR/blog/cipher/avaliacao-correcao-vulnerabilidades-seguranca-dados>. Acesso em: 17/02/2025.

7 IBM. **O que é o ciclo de vida do gerenciamento de vulnerabilidades?** 2023. Disponível em: <https://www.ibm.com/br-pt/think/topics/vulnerability-management-lifecycle>. Acesso em: 29/09/2023.

8 RESCORLA, E. Is finding security holes a good idea? **IEEE Security & Privacy**, IEEE Computer Society, Los Alamitos, CA, USA, v. 3, n. 01, p. 14–19, jan. 2005. ISSN 1558-4046. Disponível em: <https://doi.ieeecomputersociety.org/10.1109/MSP.2005.17>.

9 JANSON, F. **Ubuntu, debian, fedora, redhat and opensuse : a comparison in cve on linux distributions**. 2018. 35 p. OAI: oai:DiVA.org:his-15244. Disponível em: <https://urn.kb.se/resolve?urn=urn:nbn:se:his:diva-15244>. Acesso em: 28/02/2024.

10 PIANTADOSI, V. et al. Fixing of security vulnerabilities in open source projects: a case study of apache http server and apache tomcat. In: **2019 12th IEEE Conference on Software Testing, Validation and Verification (ICST)**. [S.l.: s.n.], 2019. p. 68–78.

11 ALFADEL, M. et al. Empirical analysis of security vulnerabilities in python packages. In: **2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)**. [S.l.: s.n.], 2021. p. 446–457.

12 LIN, J. et al. Vulnerability management in linux distributions: an empirical study on debian and fedora. **Empirical Software Engineering**, Springer, v. 28, n. 2, p. 47, 2023.

13 WU, Q. et al. Os-aware vulnerability prioritization via differential severity analysis. In: **31st USENIX Security Symposium (USENIX Security 22)**. Boston, MA: USENIX Association, 2022. p. 395–412. ISBN 978-1-939133-31-1. Disponível em: <https://www.usenix.org/conference/usenixsecurity22/presentation/wu-qiushi>. Acesso em: 15/06/2024.

14 FREI, S. et al. Large-scale vulnerability analysis. In: **Proceedings of the 2006 SIGCOMM Workshop on Large-Scale Attack Defense**. New York, NY, USA: Association for Computing Machinery, 2006. (LSAD '06), p. 131–138. ISBN 1595935711. Disponível em: <https://doi.org/10.1145/1162666.1162671>.

15 NAYAK, K. et al. Some vulnerabilities are different than others. In: STAVROU, A. et al. (Ed.). **Research in Attacks, Intrusions and Defenses**. Cham: Springer International Publishing, 2014. p. 426–446. ISBN 978-3-319-11379-1.

16 OZMENT, A.; SCHECHTER, S. E. Milk or wine: does software security improve with age? In: **15th USENIX Security Symposium (USENIX Security 06)**. Vancouver, B.C. Canada: USENIX Association, 2006. Disponível em: <https://www.usenix.org/conference/15th-usenix-security-symposium/milk-or-wine-does-software-security-improve-age>.

17 EDWARDS, N.; CHEN, L. An historical examination of open source releases and their vulnerabilities. In: **Proceedings of the 2012 ACM Conference on Computer and Communications Security**. New York, NY, USA: Association for Computing Machinery, 2012. (CCS '12), p. 183–194. ISBN 9781450316514. Disponível em: <https://doi.org/10.1145/2382196.2382218>.

18 BERES, Y. et al. Analysing the performance of security solutions to reduce vulnerability exposure window. In: **2008 Annual Computer Security Applications Conference (ACSAC)**. [S.l.: s.n.], 2008. p. 33–42.

19 REDHAT. **Product documentation for red hat security data api 1.0**. 2024. Disponível em: <https://docs.redhat.com/en/documentation/red_hat_security_data_api/1.0>. Acesso em: 31/08/2024.

20 ISO/IEC. **Information technology – security techniques – vulnerability disclosure**. [S.l.]: ISO.

21 WALSHE, T.; SIMPSON, A. Coordinated vulnerability disclosure programme effectiveness: issues and recommendations. **Computers & Security**, v. 123, p. 102936, 2022. ISSN 0167-4048. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0167404822003285>.

22 NIST. **National vulnerability database**. 2022. Disponível em: <https://nvd.nist.gov>. Acesso em: 31/08/2024.

23 NIST. **Search vulnerability database**. Disponível em: <https://nvd.nist.gov/vuln/search>. Acesso em: 17/08/2025.

24 NVDLIB. **Documentation**. 2024. Disponível em: <https://nvdlib.com/>. Acesso em: 05/08/2024.

25 NIST. **Developes**. 2022. Disponível em: <https://nvd.nist.gov/developers/vulnerabilities>. Acesso em: 17/08/2025.

26 NIST. **Request an api key**. Disponível em: <https://nvd.nist.gov/developers/request-an-api-key>. Acesso em: 19/11/2023.

27 ALMALINUX. **About almaLinux wiki**. 2023. Disponível em: <https://wiki.almalinux.org>. Acesso em: 09/09/2023.

28 ROCKYLINUX. **Keeping open source open**. 2023. Disponível em: <https://rockylinux.org/news/keeping-open-source-open>. Acesso em: 09/09/2023.

29 REDHAT. **Explaining red hat errata (RHSA, RHBA, and RHEA)**. 2024. Disponível em: <https://access.redhat.com/articles/explaining_redhat_errata>. Acesso em: 01/04/2024.

30 REDHAT. **cve.json**. Disponível em: <https://access.redhat.com/hydra/rest/securitydata/cve.json>. Acesso em: 23/06/2024.

31 ALMALINUX. **Security measures**. 2024. Disponível em: <https://almalinux.org/security>. Acesso em: 31/08/2024.

32 ALMALINUX. **AlmaLinux errata - about errata**. 2024. Disponível em: <https://wiki.almalinux.org/documentation/errata.html>. Acesso em: 31/08/2024.

33 ALMALINUX. **AlmaLinux 8 - errata.full.json**. 2025. Disponível em: <https://errata.almalinux.org/8/errata.full.json>. Acesso em: 14/07/2024.

34 ALMALINUX. **AlmaLinux 9 - errata.full.json**. 2025. Disponível em: <https://errata.almalinux.org/9/errata.full.json>. Acesso em: 14/07/2024.

35 ROCKYLINUX. **Rocky linux errata**. 2024. Disponível em: <https://wiki.rockylinux.org/rocky/errata>. Acesso em: 19/09/2024.

36 ROCKYLINUX. **apollo.openapi.json**. Disponível em: <https://apollo.build.resf.org/v2/advisories?page=0>. Acesso em: 21/09/2024.

37 DEBIAN. **Security bug tracker**. 2024. Disponível em: <https://security-tracker.debian.org/tracker>. Acesso em: 15/02/2024.

38 DEBIAN. **json**. 2024. Disponível em: <https://security-tracker.debian.org/tracker/data/json>. Acesso em: 15/02/2024.

39 DEBIAN. **Suporte a usuários(as)**. 2024. Disponível em: <https://www.debian.org/support.pt.html>. Acesso em: 15/12/2024.

40 DEBIAN. **security-tracker**. 2025. <https://salsa.debian.org/security-tracker-team/security-tracker>. [Online: acessado em 13/10/2025].

41 UBUNTU. **Security**. 2024. Disponível em: <https://ubuntu.com/security>. Acesso em: 01/09/2024.

42 UBUNTU. **Ubuntu security notices**. 2024. Disponível em: <https://ubuntu.com/security/notices>. Acesso em: 01/09/2024.

43 UBUNTU. **json**. 2024. Disponível em: <https://ubuntu.com/security/cves.json>. Acesso em: 01/09/2024.

44 UBUNTU. **CVE-2021-3928**. 2021. Disponível em: <https://ubuntu.com/security/CVE-2021-3928>. Acesso em: 19/04/2025.

45 CANONICAL. **ubuntu-com-security-api**. [S.l.], 2025. Disponível em: <https://github.com/canonical/ubuntu-com-security-api>. Acesso em: 13/10/2025.

46 REDHAT. **What to know about CentOS Linux EOL**. 2024. <https://www.redhat.com/en/topics/linux/centos-linux-eol>. [Online: acessado em 08/09/2025].

47 ROCKYLINUX. **História**. 2025. <https://rockylinux.org/pt-BR/about>. [Online: acessado em 08/09/2025].

48 ALMALINUX. **AlmaLinux is Born!** 2021. <https://blog.cloudlinux.com/almalinux-is-born>. [Online: acessado em 08/09/2025].

49 CANONICAL. **Ubuntu Pro**. [S.l.], 2025. Disponível em: <https://ubuntu.com/engage/pt/ubuntu-pro>. Acesso em: 04/10/2025.

50 NIST. **Vulnerability metrics**. 2022. <https://nvd.nist.gov/vuln-metrics/cvss>. [Online: acessado em 31/08/2024].

51 REDHAT. **Severity ratings**. 2024. <https://access.redhat.com/security/updates/classification>. [Online: acessado em 13/09/2024].

52 MURRAY, A. **Securing open source through cve prioritisation**. 2023. <https://ubuntu.com/blog/securing-open-source-through-cve-prioritisation>. [Online: acessado em 01/09/2024].

53 DEBIAN. **Debian security tracker – severity levels**. 2025. <https://security-team.debian.org/security_tracker.html#severity-levels>. [Online: acessado em 01/09/2024].

54 UBUNTU. **Upstream**. [S.l.], 2017. Disponível em: <https://wiki.ubuntu.com/Upstream>. Acesso em: 29/09/2025.

55 CATTELAIN, G. **Novidade: Lançamento do Red Hat Enterprise Linux 9**. 2022. <https://www.redhat.com/pt-br/blog/hot-presses-red-hat-enterprise-linux-9>. [Online: acessado em 07/10/2025].

56 CANONICAL. **The Ubuntu lifecycle and release cadence**. [S.l.], 2025. Disponível em: <https://ubuntu.com/about/release-cycle>. Acesso em: 07/10/2025.

57 UBUNTU. **Ubuntu pro**. [S.l.], 2025. Disponível em: <https://ubuntu.com/pro>. Acesso em: 07/03/2025.