Hashing de similaridade estrutural e por bytes em binários ELF de distribuições Linux: uma análise experimental comparativa

Felipe Duarte
Universidade Federal do Paraná
Curitiba, Paraná, BR
felipeduarte@ufpr.br

Marco Antonio Zanata Alves
Universidade Federal do Paraná
Curitiba, Paraná, BR
mazalves@ufpr.br

João Pincovscy
Centro de Pesquisa e Desenvolvimento para a Segurança das Comunicações
Brasília, Distrito Federal, BR
pincovscy@cepesc.gov.br

Paulo Lisboa de Almeida
Universidade Federal do Paraná
Curitiba, Paraná, BR
paulorla@ufpr.br

André Grégio
Universidade Federal do Paraná
Curitiba, Paraná, BR
gregio@ufpr.br

ABSTRACT

This work presents a systematic evaluation of four similarity hashing techniques applied to ELF binaries compiled in three Linux distributions. Two structural, opcode-based approaches (MinHash and SimHash) and two widely used byte-based techniques (ssdeep and TLSH) are compared under real recompilation scenarios. The analysis includes global statistics, intra- and inter-family separation, and ordinal metrics of agreement between similarity rankings. The results show that MinHash and SimHash preserve internal relationships among recompiled variants and yield relatively stable rankings, whereas ssdeep and TLSH exhibit asymmetric distributions, a predominance of near-zero similarities, and limited ability to distinguish equivalent variants. These findings indicate that structural approaches are better suited for comparing recompiled ELF binaries across different distributions and system versions.

KEYWORDS

ELF, *similarity hashing*, MinHash, SimHash, ssdeep, TLSH, Linux

1 INTRODUÇÃO

Pesquisadores têm demonstrado interesse em medir a similaridade entre binários Linux no formato *Executable and Linkable Format* (ELF) [1–3]. Esse interesse é reforçado por casos recentes de *malware* que utiliza executáveis ELF em diferentes contextos: o *Cyclops Blink*, documentado por agências como NCSC e CISA em 2022 como um *malware* modular voltado a dispositivos de rede Linux [4], e relatórios mais recentes que descrevem o uso crescente de binários ELF em campanhas direcionadas a infraestruturas de nuvem [3]. Em paralelo, estudos indicam que o formato ELF é amplamente utilizado em ecossistemas heterogêneos, de dispositivos IoT a servidores em nuvem, e que a análise de sua estrutura continua relevante para compreender o comportamento de software malicioso e legítimo em múltiplas arquiteturas [1–3].

Apesar disso, boa parte das abordagens de *clustering* e *fuzzy hashing* foi originalmente desenvolvida e avaliada no contexto de executáveis *Windows Portable Executable* (PE) [5, 6], refletindo a predominância desse formato nos estudos de similaridade de binários. Essa ênfase reduz a aplicabilidade direta de muitos métodos ao ambiente Linux, cuja estrutura de arquivos difere substancialmente do formato PE [7]. Como resultado, ferramentas amplamente difundidas, como *ssdeep*, *sdhash* e *TLSH*, embora eficazes para arquivos genéricos, não exploram características específicas do formato ELF nem aspectos estruturais do código executável em sistemas Linux.

Essa assimetria tem implicações práticas. A detecção e o agrupamento de variantes de *malware* ELF ainda dependem, em grande medida, de ferramentas genéricas originalmente projetadas para arquivos arbitrários, como *ssdeep* [8], *sdhash* [9] e *TLSH* [6]. Esses algoritmos, criados entre meados dos anos 2000 a 2010, continuam amplamente utilizados em fluxos de análise e detecção [3], mas não tratam características específicas do formato ELF, o que limita sua capacidade de lidar com transformações introduzidas por recompilações, otimizações de compilador e variações específicas ao ambiente Linux. Em 2020, Mercês et al. [7] apresentaram o *telfhash*, uma das primeiras propostas explicitamente voltadas à detecção de similaridade entre binários ELF maliciosos. A proposta consistiu em extrair os nomes de funções exportadas presentes na tabela de símbolos dos binários e aplicar o algoritmo *Trend Locality-Sensitive Hash* (TLSH) [6] para gerar assinaturas comparáveis entre amostras. O *telfhash* mostrou resultados promissores na identificação e no agrupamento de famílias de *botnets* multiplataforma, demonstrando que a incorporação de metadados de funções pode aprimorar a comparação entre amostras ELF. Ainda assim, o método expôs desafios abertos, como a dificuldade em analisar binários stripped (sem símbolos) e a dependência da tabela de símbolos, que pode tornar as assinaturas sensíveis a modificações na interface exportada.

Até onde sabemos, ainda são escassas avaliações sistemáticas de diferentes técnicas de *Locality-Sensitive Hashing* (LSH) e *fuzzy hashing* aplicadas a binários ELF sob as mesmas condições experimentais, o que inclui comparar métodos de hashing baseados em opcode e métodos baseados no conteúdo bruto do binário, medindo sua estabilidade intra/inter-distribuição e seu comportamento diante de pequenas variações geradas por recompilações reais. Questões como a preservação de similaridade entre variantes recompiladas e o custo computacional associado a cada método permanecem pouco exploradas, especialmente no contexto de comparação direta entre MinHash, SimHash, TLSH e ssdeep aplicados ao mesmo conjunto de binários ELF compilados em múltiplas distribuições Linux.

Assim, este trabalho propõe uma análise comparativa de múltiplos algoritmos de hashing aplicados a binários ELF, incluindo

abordagens baseadas em *feature hashing*, *MinHash*, e *fuzzy hashes* clássicos como *TLSH* e *ssdeep* [6, 10]. O objetivo é descrever suas propriedades e limitações de forma reprodutível e contextualizada, contribuindo para uma compreensão mais precisa do estado atual das técnicas de similaridade de código executável em ambientes Linux, utilizando um *dataset* composto por binários reais de múltiplas distribuições, gerando matrizes de similaridade, análises intra/inter-família, rankings e distribuições globais que fundamentam a discussão apresentada nas seções seguintes.

2 FUNDAMENTAÇÃO TEÓRICA E TRABALHOS CORRELATOS

A medição de similaridade entre binários tem sido estudada sob diferentes perspectivas, com duas categorias de abordagens frequentemente empregadas em pesquisas recentes: o *fuzzy hashing* e o *Locality-Sensitive Hashing* (LSH).

O *fuzzy hashing* foi introduzido na forense digital para identificar arquivos quase idênticos [11] e posteriormente aprimorado para refletir semelhanças parciais por meio de fragmentação, seleção de regiões informativas e funções sensíveis à localidade [6, 12]. Essas técnicas mantêm relevância prática porque permitem comparar grandes volumes de arquivos de forma eficiente, sem necessidade de decodificação completa de seu conteúdo.

Já as abordagens baseadas em LSH surgiram de estudos sobre busca aproximada e deduplicação em bases de dados grandes [13–15], explorando funções de hash probabilísticas que preservam, com alta probabilidade, a vizinhança entre vetores em espaços de alta dimensionalidade. A adaptação desses métodos à análise de software refletiu a necessidade de identificar amostras semelhantes - inclusive de código malicioso - em contextos heterogêneos, como sistemas Linux e dispositivos IoT, nos quais o formato ELF é predominante [1, 7]. Assim, ambas as subáreas oferecem bases conceituais distintas para a medição de similaridade em binários ELF: uma derivada da comparação direta de conteúdo e outra fundamentada em projeções sensíveis à localidade.

2.1 Fuzzy Hashing

As técnicas de *fuzzy hashing* surgiram como uma extensão dos algoritmos tradicionais de resumo criptográfico, com o objetivo de quantificar similaridade parcial entre arquivos em vez de igualdade exata [11, 12]. A ideia fundamental consiste em gerar assinaturas de similaridade que preservam relações locais do conteúdo, permitindo que pequenas modificações - como inserção, remoção ou deslocamento de blocos - resultem em diferenças proporcionais na assinatura final.

O *Context Triggered Piecewise Hashing* (CTPH), proposto por Kornblum [11] e implementado no *ssdeep*, inaugurou esse paradigma ao introduzir uma segmentação adaptativa do conteúdo, calculando resumos locais que podem ser comparados de forma aproximada. Posteriormente, o *sdhash* [12] aprimorou o processo por meio de seleção de fragmentos com alta entropia, enquanto o *TLSH* [6] incorporou uma função de hash sensível à localidade para melhorar a distribuição e a comparabilidade das assinaturas. Essas abordagens são amplamente utilizadas na triagem de grandes volumes de arquivos e na análise inicial de coleções de *malware*, uma vez que permitem comparar assinaturas sem decodificar ou descompactar os executáveis.

Entretanto, a dependência direta do conteúdo binário torna o *fuzzy hashing* sensível a recompilações, otimizações de compilador e transformações que alteram *offsets* e alinhamentos de código - efeitos comuns em binários ELF. Como consequência, sua capacidade de refletir semelhança estrutural ou semântica entre programas é limitada, o que motivou o surgimento de técnicas que operam sobre representações internas mais abstratas do código.

2.2 Locality-Sensitive Hashing (LSH)

O *Locality-Sensitive Hashing* (LSH) foi desenvolvido para permitir busca aproximada e agrupamento eficiente em coleções de alta dimensionalidade, onde comparações exatas seriam inviáveis em termos de custo [13–15]. O princípio central consiste em empregar funções de hash projetadas para preservar probabilisticamente a vizinhança entre amostras: quanto mais próximos dois vetores são em um espaço métrico, maior a probabilidade de que produzam o mesmo valor de hash.

Abordagens como o *MinHash* [13] e as *random projections* [14, 15] implementam essa ideia de forma distinta, mas com o mesmo objetivo de reduzir a dimensionalidade e acelerar comparações. No contexto de análise de binários, o LSH passou a ser utilizado para representar arquivos a partir de características derivadas do código - como sequências de opcodes ou vetores de frequência de instruções - permitindo medir similaridade sem depender do conteúdo bruto [7].

A principal vantagem dessas técnicas está na escalabilidade: quando associadas a estruturas de indexação apropriadas, possibilitam busca aproximada em tempo sublinear e identificação eficiente de vizinhos mais próximos em grandes coleções. O desempenho do LSH depende de escolhas práticas de configuração, como o número de funções de hash aplicadas e o tamanho das janelas usadas para segmentar o código. Mudanças pequenas nesses parâmetros podem alterar a distribuição dos valores de hash e, com isso, modificar o número de colisões observadas entre amostras parecidas. Essa variação é ainda mais perceptível quando os binários são produzidos por *toolchains* diferentes ou por arquiteturas distintas do formato ELF, em que ajustes de alinhamento e substituições de instruções equivalentes afetam a forma como o código é representado internamente.

2.3 Síntese e Limitações Atuais

As abordagens de *fuzzy hashing* e *Locality-Sensitive Hashing* representam linhas complementares na medição de similaridade entre binários ELF. Enquanto o *fuzzy hashing* enfatiza simplicidade de aplicação e independência de formato, o LSH busca capturar similaridade estrutural por meio de representações vetoriais e funções de dispersão probabilísticas.

A literatura existente demonstra avanços em ambas as frentes, mas ainda carece de estudos que analisem, sob condições controladas, o impacto de recompilações, otimizações de compilador e pequenas mutações sobre as assinaturas resultantes. Além disso, os protocolos experimentais e métricas variam significativamente entre trabalhos no contexto de executáveis Linux.

Diante disso, trabalhos como o *telfhash* [7] reforçam a importância de incorporar propriedades específicas do formato ELF - como

metadados de símbolos e seções - na construção de assinaturas, ao mesmo tempo em que evidenciam limitações persistentes em relação a binários *stripped* e à estabilidade das assinaturas frente a modificações sutis. Por isso, realizamos neste trabalho uma análise sistemática e reprodutível de diferentes técnicas de *hashing* aplicadas a binários ELF, a fim de avaliar seu comportamento sob cenários equivalentes de teste.

3 METODOLOGIA

Esta seção descreve o protocolo experimental empregado para avaliar, sob condições controladas e reprodutíveis, diferentes técnicas de *hashing* aplicadas a binários ELF. O objetivo central é comparar métodos baseados no conteúdo bruto do arquivo, tradicionalmente associados ao *fuzzy hashing*, e métodos que operam sobre representações internas do código executável, especialmente sequências de *opcodes*. Esses grupos representam paradigmas distintos de captura de similaridade e, portanto, a metodologia foi organizada de modo a evitar comparações diretas que desconsiderem as diferenças estruturais entre as assinaturas produzidas.

3.1 Construção do Conjunto de Binários

Para garantir diversidade e variabilidade introduzida por *toolchains* distintas, foram selecionadas três distribuições Linux, sendo elas, Ubuntu, nas versões: 25.10, 24.04, 22.04; Debian (13, 12, 11) e Fedora (41, 40, 39). Em cada ambiente, todos os executáveis presentes em `/usr/bin` foram extraídos a partir de contêineres Docker padronizados, após instalação dos pacotes essenciais (`binutils`, `coreutils`, `findutils`). A lista resultante foi então filtrada para serem mantidos apenas os binários que aparecem em todas as combinações de distribuição e versão, garantindo que as diferenças observadas derivem de variações internas do binário.

3.2 Extração de Representações Internas (Opcodes)

As técnicas estruturais do Grupo A atuam sobre uma representação derivada do fluxo de instruções. Cada binário foi passado pelo processo de *disassembly* via `objdump -d`, e dessa saída foram extraídos exclusivamente os opcodes, descartando operandos e constantes imediatas. Essa sequência linear de instruções foi transformada em listas de 4-gramas, representando padrões locais de execução. Essa granularidade é consistente com abordagens anteriores em análise de mutações e semelhança de código [10]. Os 4-gramas produzem, para cada arquivo, um conjunto de tokens adequados às técnicas de LSH descritas a seguir.

3.3 Hashing Estrutural (LSH sobre 4-gramas)

Duas técnicas foram aplicadas exclusivamente a essa representação baseada em código:

3.3.1 (a) MinHash. Os conjuntos de 4-gramas foram convertidos em vetores de 512 posições por meio de funções de hash independentes. A similaridade entre duas amostras é definida como a fração de posições idênticas entre suas assinaturas, o que aproxima o índice de Jaccard dos conjuntos originais.

3.3.2 (b) SimHash. Cada 4-grama foi mapeado para um código de 64 bits gerado via *MurmurHash3*. A projeção ponderada desses códigos produz vetores de sinais cujo resultado final é comparado por meio da distância de *Hamming*, convertida em uma medida de similaridade. Essas técnicas compõem o Grupo A e compartilham a característica de abstrair o conteúdo binário em padrões de instruções, reduzindo a influência de alinhamentos ou *offsets* específicos do ELF.

3.4 Hashing Baseado no Conteúdo Bruto (*Fuzzy Hashing*)

O Grupo B utiliza os *bytes* do arquivo, sem o processo de *disassembly* ou interpretação estrutural.

3.4.1 ssdeep. Implementa o *Context Triggered Piecewise Hashing*, produzindo assinaturas textuais cujos blocos refletem transições locais ao longo do arquivo.

3.4.2 TLSH. Gera uma assinatura baseada em histogramas de *bytes* e medidas de dispersão local. A similaridade é inversamente proporcional à distância TLSH calculada entre duas amostras. Por atuarem sobre o conteúdo bruto, essa técnica captura padrões estatísticos do arquivo, mas são sensíveis a mudanças introduzidas por recompilações e reorganizações internas comuns em binários ELF.

3.5 Construção das Matrizes de Similaridade

Cada técnica produz uma matriz de similaridade $M \in \mathbb{R}^{N \times N}$, em que cada entrada M_{ij} representa a similaridade entre duas amostras do conjunto filtrado.

As dimensões variam entre grupos:
1071×1071 para MinHash e SimHash (Grupo A);
1098×1098 para ssdeep e TLSH (Grupo B).

Essas diferenças derivam dos conjuntos finais de binários disponíveis para cada pipeline. Por esse motivo, comparações diretas entre técnicas de grupos distintos não são realizadas.

Métodos como `ssdeep` e `TLSH`, utilizados neste trabalho, apresentam um comportamento diferente das técnicas estruturais baseadas em opcode. No `ssdeep`, a similaridade depende da coincidência entre segmentos delimitados por *context triggers*; pequenas alterações introduzidas por recompilações - como ajustes de alinhamento, variação de *offsets* e reorganização de seções - costumam deslocar esses pontos de disparo, resultando em valores iguais a zero para a maior parte dos pares, mesmo quando o código-fonte é semanticamente idêntico. Já o `TLSH` produz assinaturas derivadas de histogramas de *bytes* e medidas de dispersão local, que tendem a variar pouco entre versões próximas de binários pequenos, concentrando os valores de similaridade em uma faixa estreita. Esses efeitos já foram discutidos na literatura de *fuzzy hashing* [6, 11], e explicam a predominância de valores muito baixos observada nas matrizes do Grupo B.

3.6 Avaliação Intra/Inter-Família

Os métodos são avaliados quanto à capacidade de manter coesão entre binários semanticamente equivalentes compilados em distribuições diferentes. Para cada técnica são definidas:

- **Similaridade intra-família**: pares de binários com mesmo nome originados de distribuições diferentes.
- **Similaridade inter-família**: pares de binários com nomes distintos.

A separação entre essas duas distribuições permite avaliar se a técnica preserva proximidade entre variantes reais do mesmo programa.

3.7 Avaliação Baseada em Ranking

Dado que diferentes técnicas operam em escalas distintas, análises complementares baseadas na ordenação das amostras foram realizadas:

- *ranking* local por binário (ordenação dos vizinhos mais similares);
- *Top-K* agreement entre técnicas;
- correlação de Spearman e Kendall-τ entre *rankings*;
- análise qualitativa da estabilidade intra-técnica entre amostras equivalentes.

Os resultados dessas avaliações são apresentados na Seção 4.

4 AVALIAÇÃO EXPERIMENTAL

Esta seção apresenta os resultados obtidos a partir das quatro técnicas avaliadas - MinHash, SimHash, ssdeep e TLSH - aplicadas ao conjunto de binários ELF descrito na Seção 3. O objetivo não é comparar escalas heterogêneas, mas examinar o comportamento interno de cada método diante de recompilações reais, analisando: (i) estatísticas globais das matrizes de similaridade; (ii) separação intra/inter-família; e (iii) comportamento ordinal capturado por correlações e medidas de concordância entre rankings.

4.1 Estatísticas Globais das Matrizes de Similaridade

A Tabela 1 resume as estatísticas dos valores *off-diagonal* das matrizes de similaridade. As técnicas baseadas em opcode (MinHash e SimHash) exibem distribuições contínuas e bem comportadas, com concentração em regiões específicas do intervalo $[0, 1]$. No caso do `ssdeep`, é importante ressaltar que a técnica é definida na escala original (0-100), o que explica a maior variabilidade absoluta dos valores e a assimetria observada em sua distribuição. Essa escala não é diretamente comparável às medidas contínuas (normalizadas em 0-1) utilizadas por MinHash, SimHash e TLSH. TLSH apresenta valores baixos e relativamente estáveis, consistentes com sua formulação baseada em histogramas e diferenças locais de dispersão.

Tabela 1: Estatísticas globais das matrizes de similaridade

Técnica	N	Média	DP	P25	Mediana	P75
MinHash	1071	0.521	0.090	0.465	0.523	0.580
SimHash	1071	0.750	0.066	0.704	0.745	0.793
ssdeep	1098	0.622	7.499	0.000	0.000	0.000
TLSH	1098	0.011	0.006	0.007	0.010	0.013

Cada coluna da Tabela 1 descreve características centrais das distribuições de similaridade. A coluna N indica o número de pares considerados. "Média" e "DP" resumem o nível de concentração ou dispersão dos valores, enquanto P25, Mediana e P75 delimitam a região central da distribuição, evidenciando assimetria e possíveis extremos.

A Figura 1 apresenta as distribuições globais de MinHash e SimHash. As curvas apresentadas correspondem a estimativas de densidade (*KDE*), permitindo observar a forma contínua das distribuições produzidas por cada técnica. Esse tipo de visualização destaca diferenças de forma e concentração de maneira mais clara que histogramas discretos, especialmente no caso das técnicas baseadas em opcode [16]. Essa diferença decorre da maneira como cada método incorpora variações locais nas sequências de instruções.

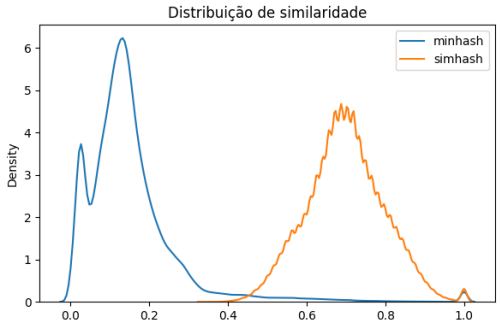

Figura 1: Densidades estimadas (*KDE*) das similaridades produzidas por MinHash e SimHash.

A Figura 1 mostra dois padrões diferentes de distribuição. O MinHash concentra-se entre 0.05 e 0.30, com múltiplos picos e máximo próximo de 0.15, refletindo variação mais ampla entre as recompilações. Já o SimHash apresenta uma distribuição mais compacta, situada entre 0.55 e 0.80, com pico em torno de 0.70, indicando maior consistência na agregação dos 4-gramas. A separação entre as curvas mostra que, embora aplicados sobre o mesmo conjunto de instruções, as técnicas capturam aspectos estruturais distintos dos binários.

4.2 Separação Intra/Inter-Família

A separação entre pares intra-família (mesmo binário compilado em distribuições distintas) e inter-família (binários diferentes) foi avaliada para cada técnica. A Tabela 2 mostra os resultados para ssdeep e TLSH - os métodos baseados em bytes.

Tabela 2: Comparação entre similaridades intra-família e inter-família para ssdeep e TLSH.

Técnica	Intra-família			Inter-família		
	n	Média	Mediana	n	Média	Mediana
ssdeep	6767	3.365	1.000	601486	0.602	0.000
TLSH	6767	0.011	0.011	601486	0.010	0.010

A Figura 2 exibe a separação gráfica entre intra e inter-família. No caso do `ssdeep`, os valores elevados presentes na região intra-família não representam aproximação estrutural entre os binários. Eles decorrem do próprio mecanismo do CTPH, que pode produzir coincidências locais entre blocos não relacionados, gerando similaridade aparente mesmo entre arquivos semanticamente distintos. Por

isso, embora a média intra seja superior à inter, a mediana permanece igual a zero, indicando que a técnica não estabelece separação consistente entre variantes compiladas do mesmo programa. No TLSH, a diferença é mais sutil, porém consistente: valores intra são ligeiramente superiores aos inter, ainda que ambos operem próximos de zero.

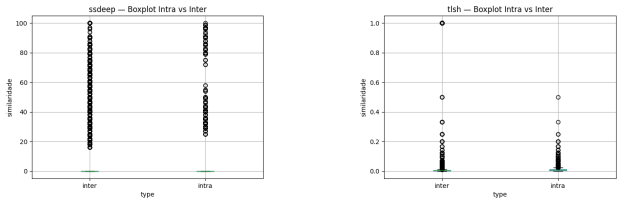

Figura 2: Comparação entre distribuições intra-família e inter-família para ssdeep e TLSH.

Esses resultados mostram que as técnicas baseadas em bytes sofrem impacto direto das alterações estruturais introduzidas por recompilações - mudanças de alinhamento, substituições de instruções equivalentes, expansão e contração de seções - o que explica a sobreposição entre as distribuições intra e inter-família.

A Tabela 3 apresenta os resultados da mesma análise para as técnicas estruturais. Diferentemente dos métodos baseados em *bytes*, MinHash e SimHash exibem separação nítida entre os pares intra e inter-família. Em ambos os casos, as similaridades intra-família são substancialmente superiores às inter-família, refletindo a capacidade dessas técnicas de preservar relações estruturais entre variantes recompiladas de um mesmo programa. No MinHash, a mediana intra (0.285) supera de forma consistente a mediana inter (0.131). O SimHash apresenta comportamento semelhante, com mediana intra de 0.734 contra 0.703 nos pares inter, ainda que a diferença entre as distribuições seja menos evidente. Esses resultados reforçam o padrão observado nas distribuições globais: as técnicas baseadas em opcode mantêm coesão interna entre amostras semanticamente equivalentes, mesmo diante das variações introduzidas por recompilações reais.

Tabela 3: Comparação entre similaridades intra-família e inter-família para MinHash e SimHash.

Técnica	Intra-família			Inter-família		
	n	Média	Mediana	*n*	Média	Mediana
MinHash	4284	0.301	0.285	1137402	0.152	0.131
SimHash	4284	0.756	0.734	1137402	0.698	0.703

4.3 Avaliação Baseada em Rankings

Para evitar dependência direta de escalas, utilizamos principalmente métricas ordinais, calculadas sobre as matrizes achatadas (pares únicos $i < j$), complementadas pelo coeficiente de correlação linear de Pearson. A Tabela 4 apresenta coeficientes de correlação (Pearson(P), Spearman(S), Kendall(K)), além de medidas de concordância Top-500(T500) e Jaccard@500(J).

Tabela 4: Correlação entre técnicas e concordância de rankings. Top-K e Jaccard@K são mostrados para $K = 500$.

Par de técnicas	P	S	K	T500	J
MinHash vs. SimHash	0.535	0.444	0.315	0.306	0.181
ssdeep vs. TLSH	0.932	0.133	0.109	0.856	0.748

MinHash vs. SimHash (Grupo A).. As correlações de Spearman (0.44) e Kendall (0.31) indicam alinhamento parcial entre as ordenações, o que é coerente com o fato de ambos operarem sobre os mesmos 4-gramas, mas com mecanismos distintos de agregação. A concordância Top-500 (0.306) reforça que os dois métodos identificam subconjuntos comuns de vizinhos mais próximos, apesar de diferenças na granularidade estrutural.

ssdeep vs. TLSH (Grupo B).. A correlação de Pearson elevada (0.93) decorre da predominância de valores próximos de zero nas duas técnicas. Entretanto, as correlações ordinais são baixas (Spearman 0.13, Kendall 0.10), indicando que a ordenação fina entre pares não é estável. A elevada concordância Top-500 (0.856) resulta da forte concentração de valores mínimos compartilhados por ambas as técnicas, e não um alinhamento estrutural entre vizinhanças relevantes.

4.4 Síntese dos Resultados

Em resumo, os resultados desta seção mostram que as técnicas baseadas em opcode (MinHash e SimHash) produzem distribuições densas e diferenciadas, com comportamento consistente frente a recompilações, enquanto os métodos baseados em bytes (ssdeep e TLSH) apresentam forte assimetria e limitada capacidade de separar variantes equivalentes, mesmo em cenários controlados. As análises de ranking indicam convergência parcial no Grupo A e convergência superficial no Grupo B, decorrente da predominância de valores nulos nas técnicas baseadas em *bytes*. Essas observações fundamentam as discussões apresentadas na Seção 6, particularmente sobre a adequação de cada técnica a cenários que envolvem recompilações, mutações leves e análises de grande escala.

5 ESTUDO DE CASO: *CLUSTERIZAÇÃO* DE AMOSTRAS ELF EQUIVALENTES

Esta seção examina, em maior detalhe, o comportamento das técnicas avaliadas ao considerar subconjuntos específicos do dataset. Selecionamos quatro programas amplamente presentes em distribuições Linux - `ls`, `cat`, `grep` e `find` - cada um representado por múltiplas versões provenientes de diferentes distribuições e versões do sistema operacional. O objetivo é verificar se as técnicas mantêm coesão entre variantes recompiladas de um mesmo programa quando analisadas isoladamente, complementando os resultados agregados apresentados na Seção 4.

5.1 Procedimento

Para cada nome de programa, foram selecionadas todas as instâncias correspondentes nos conjuntos de dados utilizados por cada técnica. As similaridades intra-grupo (pares referentes ao mesmo programa) foram então comparadas à distribuição global de cada método. Essa

análise permite observar se os padrões identificados na avaliação global se manifestam de forma consistente em casos concretos de recompilações reais.

Não são apresentados valores absolutos, pois o interesse está em compreender tendências estruturais, já que cada técnica opera em escalas distintas.

5.2 Resultados por Programa

`ls`. MinHash e SimHash atribuem similaridades intra-grupo compatíveis com a faixa densa observada nas distribuições globais, mantendo ordenações internas estáveis entre variantes recompiladas. Isso reflete a preservação de padrões locais de instrução no código do utilitário. Por outro lado, `ssdeep` raramente produz valores não nulos para essas variantes, e o `TLSH` posiciona quase todos os pares na região basal, reproduzindo exatamente o comportamento já observado na análise intra/inter-família.

`cat`. As duas técnicas baseadas em opcode mantêm agrupamento natural entre variantes do `cat`, embora com dispersão entre compilações diferentes. SimHash tende a atribuir valores mais elevados e concentrados, enquanto MinHash distribui as similaridades de maneira mais granular. Nos métodos baseados em bytes, o padrão permanece inalterado: `ssdeep` apresenta valores pontuais altos sem consistência entre pares, ao passo que TLSH produz uma faixa estreita e estável, mas com pouca separação entre amostras equivalentes e não relacionadas.

`grep`. Por ser um binário mais volumoso, o `grep` fornece um caso interessante: MinHash e SimHash mantêm coesão intra-grupo mesmo diante de maior complexidade estrutural. A estabilidade observada nas vizinhanças mais próximas confirma a capacidade dos métodos estruturais de reconhecer padrões de instrução robustos a recompilações. Já `ssdeep` produz a distribuição altamente assimétrica característica da técnica, com predominância de zeros, enquanto TLSH mantém o mesmo padrão homogêneo observado nos demais programas.

`find`. As técnicas estruturais demonstram comportamento semelhante ao observado nos demais programas: proximidade interna estável e rankings consistentes entre variantes. Tanto MinHash quanto SimHash distribuem as variantes recompiladas em regiões densas da matriz, preservando coesão. Em contraste, `ssdeep` e TLSH não apresentam separação clara entre variantes do `find` e binários não relacionados, reforçando os achados globais sobre sua sensibilidade a alterações no layout dos bytes.

5.3 Síntese do Estudo de Caso

Os quatro programas analisados ilustram, em escala reduzida, os mesmos fenômenos identificados na avaliação global da Seção 4.4. Em todos eles, as técnicas baseadas em opcode preservam relações entre variantes recompiladas, mantendo comportamento consistente mesmo em binários com estruturas e tamanhos distintos. Já o `ssdeep` apresenta apenas coincidências pontuais, sem estabilidade estrutural entre variantes, enquanto o TLSH produz assinaturas estáveis na escala global, mas insuficientes para separar, de maneira significativa, variantes equivalentes de binários distintos. Em conjunto, esses resultados corroboram que MinHash e SimHash capturam propriedades internas do código robustas a recompilações, ao passo que os métodos baseados em *bytes* permanecem fortemente dependentes do *layout* específico do arquivo, em coerência com as conclusões apresentadas na Seção 4.

6 DISCUSSÃO, LIMITAÇÕES E TRABALHOS FUTUROS

Os resultados apresentados nas seções anteriores revelam diferenças estruturais importantes entre as técnicas avaliadas e permitem discutir tanto sua adequação ao contexto de análise de binários ELF quanto as limitações persistentes na literatura. Esta discussão articula os achados experimentais com trabalhos relacionados em análise de similaridade, especialmente aqueles focados em malware multiplataforma e variantes recompiladas.

6.1 Interpretação dos Resultados

As técnicas baseadas em opcode (MinHash e SimHash) demonstraram comportamento consistente ao longo de todas as avaliações: distribuições densas e contínuas, separação clara entre variantes recompiladas e binários não relacionados, e correlações moderadas em suas ordenações internas. Esses resultados indicam que abordagens estruturais capturam propriedades do código que se mantêm estáveis mesmo quando diferentes toolchains introduzem alterações no layout do ELF - otimizações, realocação de blocos, substituições instrucionais e ajustes de alinhamento.

Essa propriedade é coerente com estudos que exploram n-grams de instruções e projeções sensíveis à localidade em cenários de mutações leves, como MutantX-S [10], nos quais padrões de fluxo de execução tendem a permanecer detectáveis mesmo após transformações sintáticas. O comportamento observado aqui reforça essa conclusão em um ambiente estritamente Linux, com binários reais oriundos de múltiplas distribuições.

Em contraste, as técnicas baseadas em bytes (ssdeep e TLSH) apresentaram assimetria acentuada e pouca capacidade discriminativa em ambientes recompilados. A predominância de similaridades nulas, mesmo em pares intra-família, indica que pequenas reorganizações no arquivo - comuns em compilações distintas - são suficientes para deslocar suas assinaturas para regiões praticamente indistinguíveis. Esse fenômeno é recorrentemente apontado na literatura de *fuzzy hashing* [17, 18], sendo um dos motivos para que propostas como o telfhash [7] busquem incorporar metadados específicos do ELF para reduzir a sensibilidade ao layout binário.

Os resultados sugerem que, embora ssdeep e TLSH permaneçam úteis para triagem de grandes coleções de arquivos heterogêneos, seu poder discriminativo no cenário Linux recompilado é limitado. Em aplicações práticas, como triagem de malware em larga escala ou agrupamento de variantes dentro de famílias conhecidas, isso implica risco elevado de falsos negativos.

6.2 Limitações do Estudo

Apesar da consistência dos resultados, algumas limitações devem ser consideradas:

- **Conjuntos distintos por técnica.** As técnicas baseadas em opcode exigem desmontagem bem-sucedida; por isso, o conjunto final de amostras não coincide exatamente com o das técnicas baseadas em bytes. Embora isso não afete a

validade das análises internas, impede comparações diretas entre grupos.
- **Ausência de mutações artificiais controladas.** Este estudo emprega apenas recompilações reais como fonte de variação. Trabalhos como MutantX-S utilizam mutações sintéticas para mapear limites de robustez. A ausência desse componente restringe a análise ao espaço de variações produzidas por toolchains reais.
- **Representação fixa (4-gramas).** Embora eficaz, o uso de 4-gramas pode não capturar padrões mais amplos de fluxo de controle. Outras granularidades poderiam revelar maior complementaridade entre MinHash e SimHash.
- **Sem avaliação sobre malware real.** Embora o foco seja a análise estrutural de binários ELF, o estudo não inclui famílias maliciosas reais. Assim, a extrapolação direta para ambientes adversariais deve ser feita com cautela.

6.3 Implicações para Análise de Malware ELF

Os achados têm implicações diretas para *pipelines* de detecção e agrupamento de malware Linux:

- Métodos baseados em bytes tendem a colapsar variantes de uma mesma família em regiões indistinguíveis, dificultando a identificação de campanhas relacionadas.
- Técnicas estruturais capturam características mais robustas a recompilações, aproximando-se das necessidades de sistemas que acompanham evolução de famílias ao longo do tempo.
- O alinhamento parcial entre MinHash e SimHash sugere que técnicas probabilísticas com diferentes mecanismos de agregação podem ser combinadas para aumentar a sensibilidade sem comprometer escalabilidade.

Essas propriedades são particularmente relevantes em cenários de resposta a incidentes, em que analistas precisam agrupar rapidamente amostras de campanhas contínuas e detectar pequenas variações introduzidas por operadores de malware.

6.4 Trabalhos Futuros

Os resultados obtidos apontam para diferentes direções de pesquisa:

- **Métodos híbridos opcode + estrutura ELF.** A incorporação de seções, símbolos remanescentes e metadados estruturais - como no telfhash - pode ampliar a capacidade discriminativa sem recorrer ao conteúdo bruto.
- **Granularidades adaptativas.** Investigar variações na representação por *n-grams* (tamanhos variáveis, janelas condicionais, filtros semânticos) pode melhorar a sensibilidade a padrões mais extensos de execução.
- **Avaliação em *datasets* maliciosos.** Replicar o experimento em coleções reais de malware ELF permitiria medir até que ponto essas conclusões se estendem para ambientes adversariais, bem como verificar robustez a ofuscação leve.
- **Estudos de mutações sintéticas.** Complementar recompilações reais com mutações controladas, seguindo metodologias de trabalhos como MutantX-S, pode fornecer uma curva mais precisa da tolerância estrutural de cada técnica.
- **Modelos de vizinhança estável.** Os resultados de ranking indicam que certas estruturas de vizinhança persistem em técnicas baseadas em opcode; explorar essas vizinhanças como grafos pode oferecer medidas adicionais de robustez e agrupamento.

6.5 Síntese

As análises desta seção podem ser resumidas em dois pontos principais. Primeiro, as técnicas baseadas em opcode (MinHash e SimHash) mantêm similaridades estáveis entre variantes recompiladas e, em conjunto com as métricas de ranking, mostram capacidade consistente de organizar binários ELF de forma coerente com sua estrutura interna. Segundo, os métodos baseados em *bytes* (ssdeep e TLSH) permanecem fortemente dependentes do *layout* do arquivo, exibindo distribuições assimétricas, sobreposição entre pares intra e inter-família e convergência superficial nas vizinhanças mais próximas. A próxima seção retoma esses resultados em perspectiva mais ampla, destacando as implicações práticas e caminhos promissores para trabalhos futuros.

7 CONCLUSÃO

Este trabalho apresentou uma avaliação sistemática de diferentes técnicas de *hashing* aplicadas a binários ELF compilados em múltiplas distribuições Linux. A análise conjunta das matrizes de similaridade, das distribuições intra/inter-família, dos rankings globais e do estudo de caso mostrou que os dois paradigmas considerados - *hashing* estrutural baseado em *opcode* e *hashing* baseado em *bytes* - produzem comportamentos distintos e, em muitos aspectos, complementares.

As técnicas estruturais (MinHash e SimHash), ao operarem sobre sequências de instruções, preservaram relações internas entre variantes recompiladas de um mesmo programa, mesmo quando o código sofreu alterações decorrentes de *toolchains* e versões de distribuição distintas. Esse comportamento se refletiu tanto nas distribuições globais quanto nas análises de vizinhança e nos quatro estudos de caso, indicando que abordagens baseadas em *opcode* capturam propriedades estruturais fortemente correlacionadas com a semântica do programa que permanecem estáveis sob recompilações legítimas.

Em contraste, as técnicas baseadas em *bytes* (ssdeep e TLSH) exibiram forte assimetria e baixa capacidade de separar variantes equivalentes, mesmo em cenários controlados. O predomínio de valores nulos no ssdeep e a pouca variação do TLSH indicam que mudanças comuns no *layout* do arquivo são suficientes para alterar significativamente suas assinaturas, reduzindo sua utilidade na comparação entre versões próximas de um binário ELF. Ainda assim, a consistência estatística observada em TLSH sugere que ele captura aspectos globais do arquivo que podem ser úteis em tarefas de triagem, embora insuficientes para discriminação fina entre variantes.

Apesar do conjunto de experimentos ser abrangente, o estudo opera em um cenário controlado e apresenta limitações naturais: não foram consideradas técnicas de ofuscação, *packers*, compressão, transformações maliciosas ou arquiteturas alternativas; tampouco avaliamos técnicas profundas de reconstrução de semântica ou representações baseadas em grafos. Além disso, o *pipeline* depende

da extração bem-sucedida de *opcodes*, o que pode falhar em binários corrompidos, altamente modificados ou intencionalmente manipulados.

Os resultados obtidos sugerem direções promissoras para trabalhos futuros. Uma possibilidade é o desenvolvimento de métodos híbridos que combinem características sintáticas (*bytes*) e estruturais (*opcodes*), de forma análoga ao que o *telfhash* [7] propôs para símbolos, mas com maior robustez a recompilações. Outra direção é estender o *pipeline* a coleções reais de malware ELF, incluindo famílias de *botnets*, *loaders* e artefatos *IoT*, em que pequenas mutações e recompilações são parte central do comportamento adversarial. Por fim, abordagens baseadas em aprendizado profundo e modelos de representação de código podem oferecer caminhos adicionais para capturar regularidades estruturais ausentes tanto no *fuzzy hashing* tradicional quanto no *LSH* clássico.

Em conjunto, os achados deste trabalho mostram que a medição de similaridade entre binários ELF exige técnicas capazes de refletir não apenas o conteúdo bruto, mas também a organização interna do código. Os resultados reforçam o papel das abordagens estruturais como alternativas mais estáveis e informativas em cenários que envolvem recompilações, mutações leves e análises em larga escala, fornecendo uma base empírica para investigações futuras sobre mecanismos de *hashing* específicos para o ecossistema ELF.

AGRADECIMENTOS

Este trabalho foi apoiado pelo Centro de Pesquisa e Desenvolvimento para a Segurança das Comunicações, bem como pelo CNPq por meio de bolsa de produtividade em pesquisa.

REFERÊNCIAS

[1] Jayanthi Ramamoorthy, Khushi Gupta, Ram C. Kafle, Narasimha K. Shashidhar, and Cihan Varol. A novel static analysis approach using system calls for linux iot malware detection. *Electronics*, 13(15):2906, 2024. doi: 10.3390/electronics13152906. URL https://doi.org/10.3390/electronics13152906.

[2] Javier Carrillo-Mondejar, Guillermo Suarez-Tangil, Andrei Costin, and Ricardo J Rodríguez. Exploring shifting patterns in recent iot malware. In *Proceedings of the European Conference on Cyber Warfare and Security*. Academic Conferences International Ltd, 2024.

[3] Nathaniel Quist and Bill Batchelor. The evolution of linux binaries in targeted cloud operations. Technical report, Unit 42 Threat Intelligence, Palo Alto Networks, 2025. Threat research report. Available at: https://unit42.paloaltonetworks.com/elf-based-malware-targets-cloud/.

[4] NCSC and CISA. Cyclops blink malware analysis report. Technical report, UK National Cyber Security Centre and US Cybersecurity and Infrastructure Security Agency, 2022. URL https://www.ncsc.gov.uk/files/Cyclops-Blink-Malware-Analysis-Report.pdf.

[5] Georg Wicherski. pehash: A novel approach to fast malware clustering. In *Proceedings of the 2nd USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET)*, 2009.

[6] Jonathan Oliver, Chun Cheng, and Yang Chen. Tlsh – a locality sensitive hash. In *2013 Fourth Cybercrime and Trustworthy Computing Workshop (CTC)*, pages 7–13, 2013. doi: 10.1109/CTC.2013.9.

[7] Bruno Mercês. Telfhash: An algorithm that finds similar malicious elf files. Technical report, Trend Micro, 2020. URL https://documents.trendmicro.com/assets/pdf/TB_Telfhash-An-Algorithm-That-Finds-Similar-Malicious-ELF-Files.pdf. Technical Brief.

[8] Vassil Roussev. ssdeep: An open source context-triggered piecewise hashing tool. Technical report, Department of Computer Science, University of New Orleans, 2009. URL https://ssdeep-project.github.io/ssdeep/index.html.

[9] Vassil Roussev. Data fingerprinting with similarity digests. In Gilbert Peterson and Sujeet Shenoi, editors, *Advances in Digital Forensics VII*, pages 207–226. Springer, 2011.

[10] Xin Hu, Kang G. Shin, Kent Griffin, and Sandeep Bhatkar. Mutantx-s: Scalable malware clustering based on static features. In *USENIX Annual Technical Conference (USENIX ATC 13)*, pages 187–198. USENIX Association, 2013.

[11] Jesse Kornblum. Identifying almost identical files using context triggered piecewise hashing. *Digital Investigation*, 3(S1):91–97, 2006. doi: 10.1016/j.diin.2006.06.015.

[12] Vassil Roussev. Data fingerprinting with similarity digests. In *IFIP International Conference on Digital Forensics VI*, pages 207–226. Springer, 2010. doi: 10.1007/978-3-642-15506-2_15.

[13] Andrei Z. Broder. On the resemblance and containment of documents. In *Proceedings of the Compression and Complexity of Sequences 1997 (SEQUENCES '97)*, pages 21–29. IEEE, 1998. doi: 10.1109/SEQUEN.1997.666900.

[14] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing (STOC)*, pages 604–613, 1998. doi: 10.1145/276698.276876.

[15] Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Similarity search in high dimensions via hashing. In *Proceedings of the 25th International Conference on Very Large Data Bases (VLDB)*, pages 518–529, 1999.

[16] Yen-Chi Chen. Histogram and kernel density estimation. https://faculty.washington.edu/yenchic/18W_425/Lec6_hist_KDE.pdf, 2018. Lecture notes, University of Washington. Accessed: 08 Dec. 2025.

[17] Fabio Pagani, Matteo Dell'Amico, and Davide Balzarotti. Beyond precision and recall: Understanding uses (and misuses) of similarity hashes in binary analysis. In *Proceedings of the 8th ACM Conference on Data and Application Security and Privacy (CODASPY)*, pages 354–365, 2018. doi: 10.1145/3176258.3176339.

[18] Lorenz Liebler and Harald Baier. Towards exact and inexact approximate matching of executable binaries. *Digital Investigation*, 30:1–15, 2019. doi: 10.1016/j.diin.2019.06.004.