

Análise Paralela de Logs SSH para Identificação de Tentativas de Força Bruta

Rodrigo Berino Pereira¹, Ricardo de la Rocha Ladeira¹

¹Instituto Federal Catarinense – Campus Blumenau
Blumenau – Santa Catarina – Brasil

rodrigo.pereira@estudantes.ifc.edu.br, ricardo.ladeira@ifc.edu.br

Abstract. *This paper presents a parallel analyzer for SSH authentication logs that counts failed authentications per source IP and ranks the addresses with the most failures, supporting the triage of potential brute-force activity. The analyzer uses a MapReduce-inspired design, memory-mapped files, POSIX threads, and thread-local hash tables. Experiments used a synthetic log with 5 million lines. The ranking matched that of a UNIX reference script, and with four threads, the median execution time dropped from 1.1603 s to 0.3845 s, yielding a 3.02 speedup. Running on a single machine, this approach provides a low-cost alternative that enables agile incident response across large volumes of logs.*

Resumo. *Este artigo apresenta um analisador paralelo para logs de autenticação SSH que conta autenticações com falha por IP de origem e classifica os endereços com mais falhas, apoiando a triagem de possíveis atividades de força bruta. O analisador combina uma abordagem inspirada no MapReduce, arquivos mapeados em memória, threads POSIX e tabelas hash locais de thread. Os experimentos usaram um log sintético com 5 milhões de linhas. A classificação correspondeu à de um script de referência UNIX e, com quatro threads, o tempo mediano de execução caiu de 1,1603 s para 0,3845 s, o que corresponde a um speedup de 3,02. Executada em uma única máquina, essa abordagem oferece uma alternativa de baixo custo que pode apoiar a resposta a incidentes de forma ágil em grandes volumes de logs.*

1. Introdução

Ataques de força bruta contra serviços SSH (*Secure Shell*) consistem em tentativas sucessivas de autenticação com diferentes credenciais e ameaçam sistemas expostos à Internet. Estudos em ambientes reais mostram que essas tentativas podem atingir grande escala e apresentar comportamento persistente e distribuído [Singh et al. 2024, Cao et al. 2019, Wu et al. 2020]. Esses eventos deixam evidências nos *logs* de autenticação, como endereço IP (*Internet Protocol*) de origem, usuário e resultado da tentativa.

A análise desses registros pode identificar fontes suspeitas e apoiar a triagem [Cao et al. 2019, Hancock et al. 2021]. No entanto, o crescimento do volume de *logs* pode atrasar essa identificação e a resposta a eventuais ataques em andamento.

Este trabalho apresenta um analisador paralelo de *logs* SSH que contabiliza falhas de autenticação por IP e produz um *ranking* de fontes potencialmente associadas a ataques de força bruta, priorizando a investigação. Embora soluções SIEM (*Security Information and Event Management*) ofereçam monitoramento em tempo real, a abordagem proposta é

uma alternativa de baixo custo, voltada ao pré-processamento e à investigação de arquivos de *log* já coletados. A paralelização reduz o tempo de processamento de grandes volumes e, com isso, o tempo até a identificação das origens a investigar, o que pode apoiar a resposta rápida a incidentes.

2. Contexto

Em serviços SSH expostos à Internet, tentativas automatizadas de força bruta podem apresentar comportamento persistente e distribuído, com múltiplas origens e grande volume de eventos de autenticação [Singh et al. 2024, Wu et al. 2020]. Nesse cenário, *logs* de autenticação permitem caracterizar esse comportamento e apoiar sua identificação e mitigação, como demonstrado em abordagens de análise de ataques SSH em larga escala [Cao et al. 2019, Hancock et al. 2021].

A quantidade de falhas de autenticação é um indicador relevante para analisar possíveis tentativas de força bruta, embora isoladamente não caracterize um ataque, especialmente em campanhas distribuídas [Singh et al. 2024, Wu et al. 2020]. Enquanto gerenciadores de *log* centralizados focam na ingestão contínua em nuvem, o reprocessamento de arquivos legados em disco exige alta vazão local. O uso de processamento paralelo em memória compartilhada reduz o tempo de processamento.

Nesse contexto, a análise de *logs* pode se beneficiar do paralelismo de dados, pois cada registro pode ser processado independentemente. O paradigma *MapReduce* organiza esse tipo de processamento em etapas de transformação e agregação [Dean and Ghemawat 2008], e trabalhos como Phoenix demonstram sua aplicação em sistemas *multicore* [Ranger et al. 2007]. A abordagem adotada neste trabalho utiliza essa ideia em memória compartilhada, com cada *thread* processando um bloco e agregando localmente as falhas antes da consolidação.

3. Metodologia

3.1. Estratégia de Análise e Validação

Para cada linha do arquivo de *log*, o analisador verifica se o registro corresponde a uma mensagem de falha do serviço *sshd*, com status *Failed* e método de autenticação suportado: *password*, *publickey* ou *keyboard-interactive/pam*. Após validar a estrutura da mensagem, extrai e normaliza o endereço de origem (IPv4 ou IPv6) e associa a ocorrência (*IP*, 1). Essa validação evita confundir um usuário chamado *from* com o delimitador do endereço IP. Além disso, as ocorrências são agregadas por IPs, que são ordenados de forma descendente por quantidade de falhas associadas e apresentados no relatório para priorizar a investigação.

A estratégia foi escolhida por simplicidade e interpretabilidade, mas possui uma limitação: elevada frequência de falhas não constitui, isoladamente, prova de atividade maliciosa. O *ranking* deve, portanto, ser interpretado como apoio à triagem.

Por fim, como parte da estratégia de validação, elaborou-se um *script* UNIX de referência, fundamentado em ferramentas de linha de comando (*grep*, *awk*, *sort* e *uniq -c*), com o objetivo de auditar e confrontar a exatidão das contagens geradas pelo analisador.

3.2. Dataset

Para avaliar o analisador de forma controlada, utilizou-se o arquivo sintético `auth_5M.log`, com cinco milhões de linhas, gerado por *script*¹. O formato reproduz o padrão das linhas de autenticação SSH do `auth.log`, contendo informações como data, hora, serviço, resultado, usuário e IP de origem (por exemplo, `Failed password for root from 185.220.101.42`), permitindo simular rastros de autenticação de sistemas reais. O gerador seleciona fontes definidas como suspeitas com probabilidade de 70%, atribuindo-lhes probabilidade de falha de 95%. As demais fontes são legítimas, com probabilidade de falha de 5%. A Tabela 1 apresenta a composição esperada.

Tabela 1. Composição esperada do dataset de cinco milhões de linhas.

Origem	Linhas	Falhas	Taxa
Fontes suspeitas	3.500.000	3.325.000	95%
Fontes legítimas	1.500.000	75.000	5%
Total	5.000.000	3.400.000	68%

O conjunto público *SSH Brute Force IP/User/Password* [Kaggle 2023] serviu de referência para a concepção do experimento por conter registros de tentativas de autenticação SSH com proporção de IPs suspeitos semelhante à adotada no gerador. Além disso, fundamentou a ideia de receber *logs* desse tipo, processá-los em paralelo e analisá-los. Como seus registros são disponibilizados em JSON, enquanto o analisador processa o formato textual do `auth.log`, a avaliação quantitativa utilizou o arquivo sintético descrito acima. O *script* gerador utiliza um conjunto de nove endereços IP possíveis, sendo cinco associados às fontes suspeitas e quatro às fontes legítimas.

3.3. Métricas de Avaliação e Procedimento Experimental

Para mensurar o desempenho, os tempos de execução foram obtidos por meio de `clock_gettime(CLOCK_MONOTONIC)`. O intervalo cronometrado abrange desde o mapeamento inicial do arquivo em memória e o particionamento dos blocos até o término da consolidação dos dados processados pelas *threads* (criação, execução e espera das tarefas). Etapas como contagem prévia, ordenação final, impressão de relatório e liberação de recursos foram desconsideradas.

Para cada configuração, realizaram-se cinco execuções, adotando-se a mediana como medida representativa do tempo de execução. A partir desses tempos, calculou-se o *speedup* ($S(p) = T_1/T_p$), a eficiência ($E(p) = S(p)/p$) e o *throughput*, expresso em milhões de linhas analisadas por segundo.

4. Arquitetura e Implementação

Implementado em linguagem C com o uso da biblioteca *POSIX Threads* (`pthread`), o analisador divide o processamento em quatro etapas: *Split*, *Map*, *Merge* e *Reduce*. Essa divisão separa o particionamento do arquivo, a identificação das falhas, a consolidação das contagens e a geração do *ranking*. A Figura 1 apresenta o fluxo dessas etapas.

No *Split*, o arquivo é mapeado em memória com `mmap()` e dividido em blocos sem interromper linhas. No *Map*, cada *thread* processa um bloco, identifica falhas e atualiza uma tabela *hash* local.

¹https://github.com/RodrigoBerino/auth_analyzer

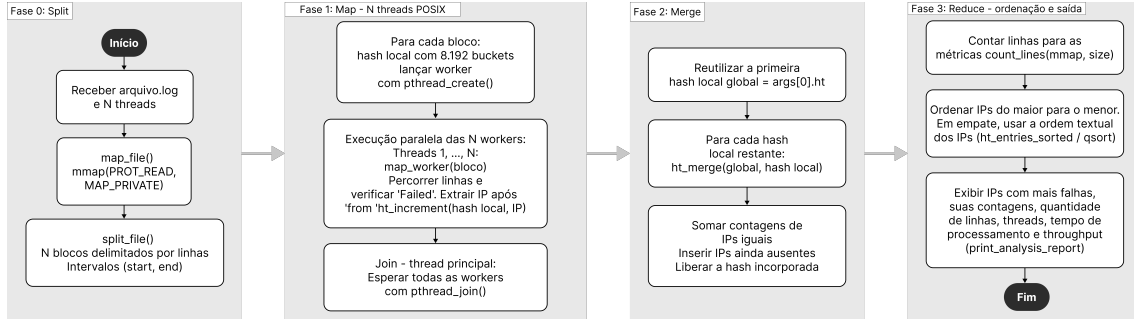


Figura 1. Arquitetura do analisador paralelo de logs SSH

Na fase *Map*, cada linha é analisada dentro dos limites do bloco atribuído à *thread*, sem alterar o conteúdo mapeado. A contagem local é incrementada somente após a validação do registro e a obtenção do endereço IP normalizado. O analisador aceita linhas terminadas em `\n` e `\r` e também o último registro do *log* sem quebra de linha.

Ao final da fase *Map*, as tabelas locais são consolidadas na fase *Merge*. A fase *Reduce* ordena os IPs pelo número de falhas. O uso de `mmap()` permite acessar o arquivo pelo espaço de endereçamento do processo, enquanto as tabelas locais reduzem a contenção [Stevens and Rago 2005, Tanenbaum 2010].

5. Avaliação Experimental

5.1. Ambiente

Os experimentos foram realizados em uma máquina com processador Intel Core i5-1335U (13ª geração, arquitetura híbrida com 2 núcleos de desempenho com *hyper-threading* e 8 núcleos de eficiência, totalizando 10 núcleos físicos e 12 processadores lógicos), 16 GB de RAM e Fedora Linux 43 Workstation, com *kernel* 7.1.13-100.fc43. O analisador foi compilado no GCC 15.3.1 com otimização `-O2` e GNU Make 4.4.1, avaliando-se configurações com 1 (*baseline*), 2, 4 e 8 *threads*.

5.2. Resultados de Segurança

Ao executar a análise no arquivo `auth_5M.log`, a diferença entre as taxas de falha das fontes suspeitas e legítimas permitiu avaliar a capacidade de recuperação das principais fontes no cenário controlado. Como esperado, os cinco IPs de origem definidos como suspeitos ocuparam as primeiras posições do *ranking*. Os resultados e o total de falhas não variaram entre as execuções com 1, 2, 4 e 8 *threads*.

Para validar os resultados obtidos pelo analisador, o *script* UNIX de referência foi executado com o mesmo arquivo (`auth_5M.log`). O resultado obtido foi idêntico: 3.399.166 falhas, distribuídas entre nove IPs. A Tabela 2 apresenta o *ranking* completo.

Esses resultados sustentam a corretude da estratégia de filtragem e agregação no cenário avaliado. Entretanto, não implicam eficácia na detecção de ataques reais, principalmente aqueles distribuídos entre vários endereços [Singh et al. 2024, Wu et al. 2020].

5.3. Resultados de Desempenho

A Tabela 3 apresenta a mediana dos tempos, o *speedup*, a eficiência e o *throughput* obtidos para 1, 2, 4 e 8 *threads*. A Figura 2 complementa a Tabela 3 ao ilustrar a diferença de tempo à medida que o número de *threads* aumenta.

Tabela 2. Ranking observado no arquivo auth.5M.1og.

IP de origem	Script com comandos UNIX	Analizador Proposto	Classe Preliminar
185.220.101.42	665.728	665.728	Suspeito
103.99.0.118	664.873	664.873	Suspeito
45.33.32.156	664.743	664.743	Suspeito
91.240.118.50	664.316	664.316	Suspeito
178.128.88.200	663.954	663.954	Suspeito
192.168.1.10	18.933	18.933	Legítimo
10.0.0.45	18.906	18.906	Legítimo
192.168.1.20	18.865	18.865	Legítimo
10.0.0.33	18.848	18.848	Legítimo
Total	3.399.166	3.399.166	

Tabela 3. Desempenho com 5 milhões de linhas.

p	Mediana dos tempos (s)	Speedup	Eficiência	Throughput (M linhas/s)
1	1,1603	1,00	1,00	4,31
2	0,6129	1,89	0,95	8,16
4	0,3845	3,02	0,75	13,00
8	0,4064	2,86	0,36	12,30

O menor tempo mediano ocorreu com quatro *threads* (0,3845 s), 3,02 vezes mais rápido que a execução com uma *thread*. O *speedup* cresceu até quatro *threads* (1,89 e 3,02), mas recuou com oito (2,86). A eficiência calculada foi decrescente (0,95, 0,75 e 0,36 para 2, 4 e 8 *threads*, respectivamente). De quatro para oito *threads*, a mediana do tempo piorou (0,3845 s e 0,4064 s) e o *throughput* diminuiu de 13 para 12,3 milhões de linhas por segundo. Neste experimento, portanto, o ganho se esgota em quatro *threads*.

6. Discussão e Limitações

A abordagem combina baixo custo e simplicidade, e pode apoiar a resposta a incidentes como camada de triagem anterior a mecanismos mais complexos. Entretanto, a frequência de falhas por IP é apenas um indicador inicial, podendo perder eficácia diante de ataques distribuídos, baixos volumes por origem e estratégias de evasão [Singh et al. 2024, Wu et al. 2020]. Já o ganho de desempenho se esgotou em quatro *threads*, sem que a causa fosse investigada. A arquitetura híbrida do processador e o *Merge* sequencial são hipóteses a analisar.

A principal ameaça à validade externa é o uso de dados sintéticos na análise quantitativa. O controle da distribuição facilita a validação, mas não reproduz a variabilidade de ambientes reais. O sistema também não bloqueia IPs suspeitos nem produz uma classificação definitiva, atuando na triagem para etapas de investigação ou resposta.

7. Conclusão

Este trabalho apresentou um analisador paralelo de *logs* SSH que agrega falhas de autenticação por endereço IP e produz um *ranking* de fontes potencialmente associadas a ataques de força bruta. A implementação combina uma arquitetura inspirada em *MapReduce*, `mmap()` e *Pthreads*, com tabelas *hash* locais para reduzir contenção.

No cenário sintético, as fontes suspeitas foram recuperadas no *ranking*, sem variação entre as configurações. Do ponto de vista da segurança, o analisador oferece uma alternativa de baixo custo, executável em uma única máquina, para triar grandes volumes de *logs* de autenticação e priorizar origens a investigar ou mitigar. Assim, pode atuar como complemento a sistemas de proteção e de monitoramento em tempo real.

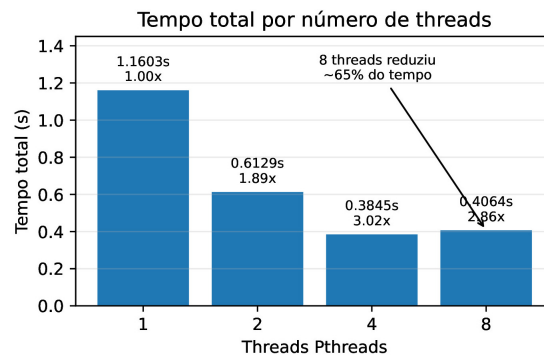


Figura 2. Tempo de execução em função do número de *threads*.

Como trabalhos futuros, pretende-se comparar os tempos obtidos com o *script* UNIX, bem como analisar soluções com ferramentas de proteção de servidores como *fail2ban* e *sshguard*. Deseja-se ainda avaliar o analisador com dados reais de autenticação e incorporar janelas temporais para correlacionar eventos, permitindo identificar ataques distribuídos.

Agradecimentos

Os autores declaram que o ChatGPT (GPT-5.6 Luna, em modo de *Thinking*) foi utilizado exclusivamente como ferramenta de apoio à revisão textual deste artigo.

Referências

- Cao, P. M., Wu, Y., Banerjee, S. S., Azoff, J., Withers, A., Kalbarczyk, Z. T., and Iyer, R. K. (2019). CAUDIT: Continuous auditing of SSH servers to mitigate brute-force attacks. *USENIX NSDI '19*, pp. 667-682.
- Dean, J. and Ghemawat, S. (2008). MapReduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1): pp. 107-113.
- Hancock, J., Khoshgoftaar, T. M., and Leevy, J. L. (2021). Detecting SSH and FTP brute force attacks in big data. *IEEE ICMLA 2021*, pp. 760-765.
- Kaggle (2023). SSH brute force IP/User/Password dataset. <https://www.kaggle.com/datasets/lako65/ssh-brute-force-ipuserpassword>.
- Ranger, C., Raghuraman, R., Penmetsa, A., Bradski, G., and Kozyrakis, C. (2007). Evaluating MapReduce for multi-core and multiprocessor systems. *HPCA '07*, pp. 13-24.
- Singh, S. K., Gautam, S., Cartier, C., Patil, S., and Ricci, R. (2024). Where the wild things are: Brute-force SSH attacks in the wild and how to stop them. *USENIX NSDI '24*, pp. 1731-1750.
- Stevens, W. R. and Rago, S. A. (2005). *Advanced Programming in the UNIX Environment*. Addison-Wesley, Upper Saddle River, NJ, USA, 2 edition.
- Tanenbaum, A. S. (2010). *Sistemas Operacionais Modernos*. Pearson Prentice Hall, São Paulo, 3 edition.
- Wu, Y., Cao, P. M., Withers, A., Kalbarczyk, Z. T., and Iyer, R. K. (2020). Mining threat intelligence from billion-scale SSH brute-force attacks. *Proceedings of the Workshop on Decentralized IoT Systems and Security*.