Objectif : Analyse des logs par des techniques d'apprentissage automatique

- Partie I : recherches préliminaires

1- État de l'art :

Mots clefs : Fouille de Données, Cloud Computing, Détection d'Anomalies, Apprentissage Profond, Flux de Données, Séries temporelles, Systèmes Distribués ;
Failure prediction, Operations, Datacenter, Machine learning, Log parsing, Online algorithm, Log analysis, Web service management, Bidirectional tree, log data, Log management, anomaly detection, AIOps, natural language processing, anomaly detection, malware, word embeddings, fastText, Anomaly Detection, Deep Learning, Log Instability, Data Quality, Intrusion detection,
Network logs, machine learning classifiers.

L'objectif comme démontré ici [16], est de transformer les logs pour avoir d'abord une vision « human-friendly » du problème et de proposer une solution. Pour cela, il faut réussir à filtrer et analyser les logs entrants.

2 types de logs : en réseau et en local, online ou offline (historique ou en temps réel).

Les premiers sont générés par des serveurs et relatent les requêtes effectuées et les ressources échangées.
Ces logs peuvent être exploités au travers d'une base de données sous forme de graphe [15] où les liens entre tables représentent les suites de messages dans les logs et donc les suites d'évènements qui pourraient conduire à une potentielle attaque. Cette approche semble par ailleurs limitée à un nombre restreint d'ordinateurs à surveiller (testé avec 45 ordinateurs). L'utilisation de la bibliothèque Java Neo4j est faite pour construire ces graphes et les utiliser ensuite dans des bdd telles que MySQL ou MongoDB.
Un autre objectif est de prévenir les intrusions sur un réseau [17]. L'analyse des échanges et la recherche de mots clefs dans ceux-ci permet de remarquer une grande différence lors d'une attaque et lors d'un usage normal :

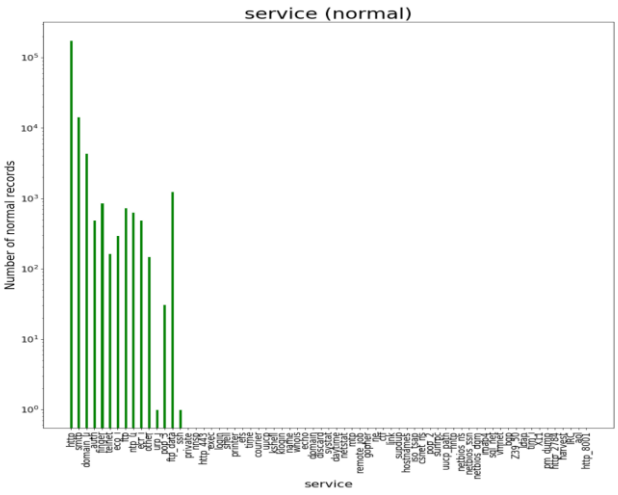


Fig. 2: Distribution of feature *service* - normal

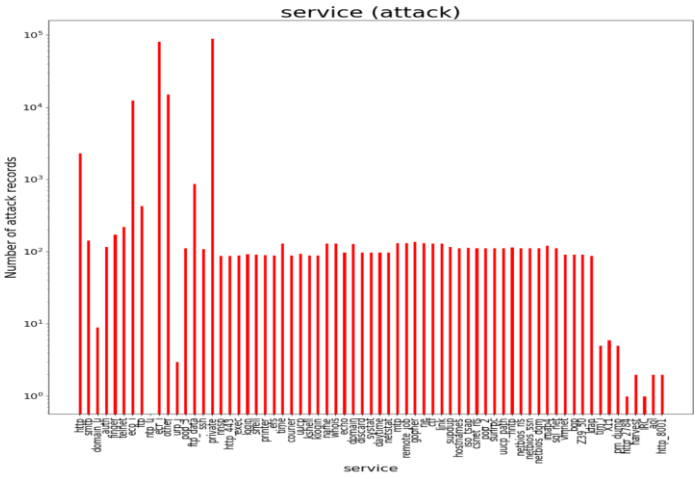


Fig. 3: Distribution of feature *service* -attack

Pour discriminer ces données, plusieurs méthodes ont été employés. Un arbre de décision, les K plus proches voisins et un réseau de neurones. Après un pré-traitement permettant de diminuer la dimension de l'étude, les K plus proches voisins donnent le meilleur résultat compte tenu de sa simplicité face aux réseaux de neurones.

Afin d'en apprendre davantage sur les logs durant une attaque, l'équipe de PWNJUTSU [19] a invité 22 Red-Teamers sur un serveur et a enregistré tous leurs mouvements afin de visualiser tous les logs durant une attaque. Toutes les entrées sont disponibles sur leur site internet, soit 16 millions de lignes de logs et 172 Gb.

Dans un premier temps, des méthodes traditionnelles ont été utilisé depuis 2007 [9], la première basée sur les SVM [18] pour détecter les défaillances sur le super-ordinateur IBM BlueGene/L. L'utilisation des SVM en [18] est rendue possible grâce à la présence d'un « niveau » d'alerte dans le log, plus le tag est faible et plus l'évènement est important. Dès lors, en conservant la suite des évènements par groupe de 5, les auteurs ont pu déterminer avec une efficacité de 73 % des séquences présentant des anomalies. Par ailleurs, cette méthode ne fonctionne que sur le set de logs utilisé dans l'étude est nécessiterait bien plus de traitement pour être généralisable telle quelle.

En 2009 [10], une méthode basée ACP a été testé sur des logs Apache Hadoop en reconstruisant des séquences de log pour détecter des anomalies dans celles-ci.
Plus tard, une nouvelle méthode [11] basée sur le Invariant Mining (IM) était testée sur les mêmes sets de logs ainsi que sur Jboss (serveur d'applications en Java). L'objectif était de regarder les relations entre les différents évènements en cherchant un invariant et reconstruire les séquences pour remarquer les anomalies, c'est-à-dire les séquences qui s'éloignent trop de l'invariant (à l'image des invariants de marquage dans un réseau de Pétri).
Ces méthodes sont assez peu efficaces et nécessitent un grand temps de calcul.
Il en existe d'autres aux résultats sensiblement similaires.

Les méthodes par Deep-Learning sont beaucoup plus récentes et efficaces.

Il existe un certain nombre d'algorithme pour analyser les logs et en faire ressortir les anomalies. Parmi eux, DeepLog [4][5] s'est annoncé en 2017 comme le meilleur, précurseur dans la matière en utilisant des réseaux de neurones et l'utilisation de « Long Short-term-memory » pour la détection.

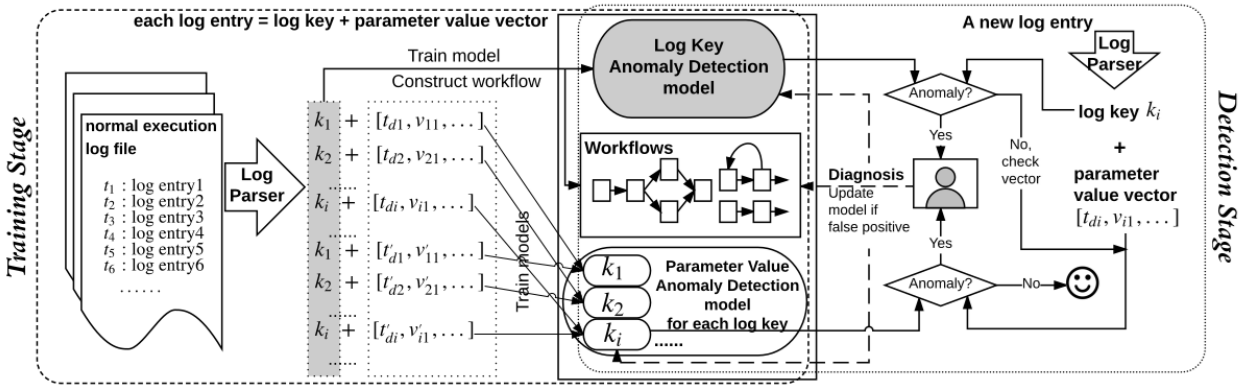


Figure 1: DeepLog architecture.

Le fonctionnement est par ailleurs assez sombre puisqu'en entrée, DeepLog prend un fichier texte ne contenant que des nombres et analyse comment ceux-ci se suivent pour prédire si la séquence suit une logique, une sémantique connue et ne présentant pas d'anomalies. Il faut par ailleurs entraîner le réseau de neurones sur un set ne présentant pas d'anomalies pour que la détection puisse se faire sur le set à analyser.
La partie de Log-Parsing est basée sur Spell [20], un algorithme qui utilise la plus longue sous-séquence pour générer le template. Ceci constitue le paramètre k_i, la suite suivante constitue le temps écoulé entre un évènement et son prédécesseur, stocké dans un vecteur.

En 2022, la même équipe a développé DeepCase [6], basé sur les mêmes concepts mais allant encore plus loin. L'approche est semi-supervisée est permet de réduire de 90 % la charge de travail d'un opérateur en n'indiquant que les évènements qu'il juge comme des anomalies, filtrant la quantité d'information affichée à un opérateur mais se base sur une base de données.

La même année, une autre équipe s'est plutôt basée sur l'utilisation de méthodes de Natural Language Processing, développant LogAnomaly [7].

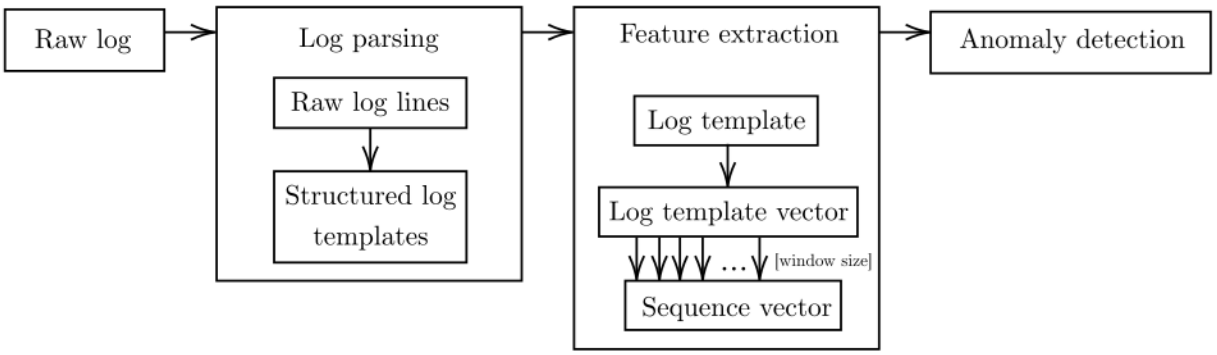


Figure 1. Model architecture for detecting anomalies in log files.

La partie de parsing (templatisation) est toujours la même mais l'extraction des features repose sur l'algorithme fastText [8], développé par Meta (Facebook).

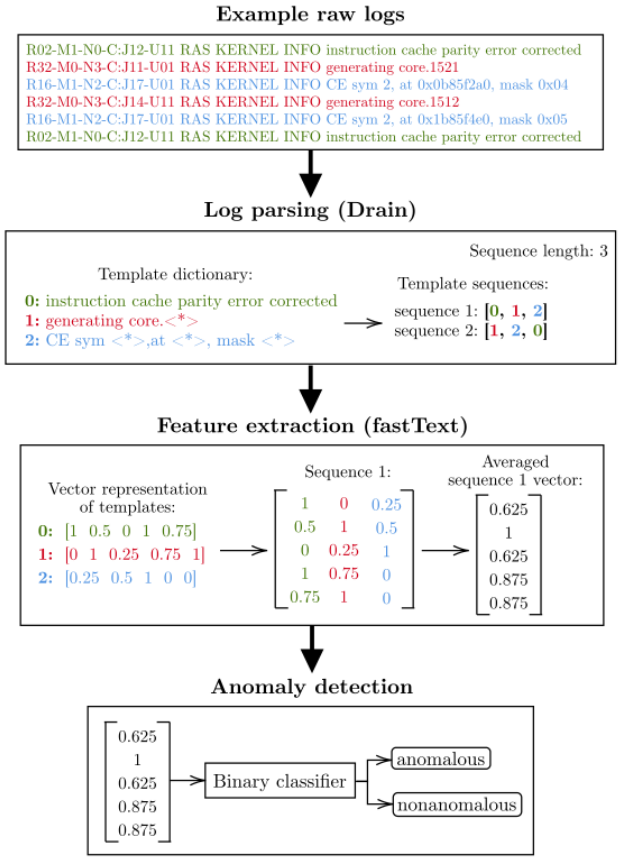


Figure 3. Scheme of detecting anomalies in system log files.

Dès lors, une classification binaire est simplement faisable. MultiLayer Perceptrons (MLP – basé sur des réseaux de neurones à plusieurs couches) est le système qui propose les meilleurs résultats avec un score F1 de 99.4 % et une accuracy de 99.4 % également, démontrant que presque toutes les anomalies sont correctement repérées par l'algorithme.
Dans cette approche, les auteurs parlent également, dans le futur, de mettre en place un apprentissage sur un set sans anomalies et une approche semi-supervisée comme ce que DeepCase propose.

On cite également LogRobust [12] qui traite les logs de la même manière que LogAnomaly, en utilisant des réseaux de neurones à LSTM bidirectionnels.

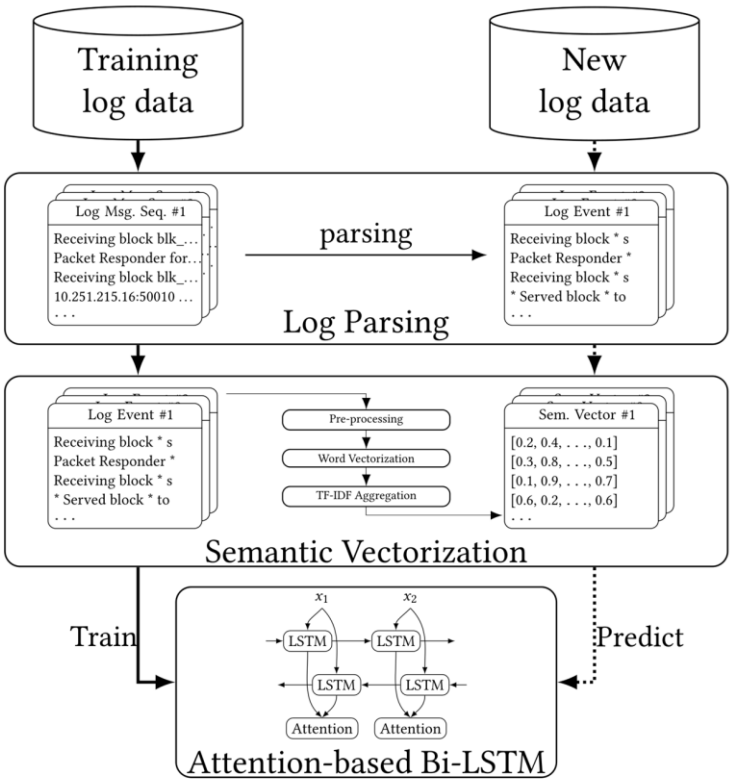


Figure 6: The Overview of LogRobust

Drain est utilisé pour la première partie, la templatisation (log-parsing).
FastText est utilisé pour vectoriser les mots et un algorithme TF-IDF (*term frequency-inverse document frequency)* s'ajoute.
Le LSTM double permet d'annuler complètement les effets secondaires dus aà la templatisation et de rendre l'algorithme très robuste.

Les auteurs de ces méthodes signalent par ailleurs une instabilité dans les logs pouvant donner des effets de bords non négligeables aux techniques employées. La solution serait de regrouper les templates générés en utilisant soit un template intermédiaire par Ft-tree (arbre à préfixe) (LogAnomaly [13]), en utilisant la sémantique du log (LogRobust) ou bien un encodeur spécifique (SwissLog [14]).

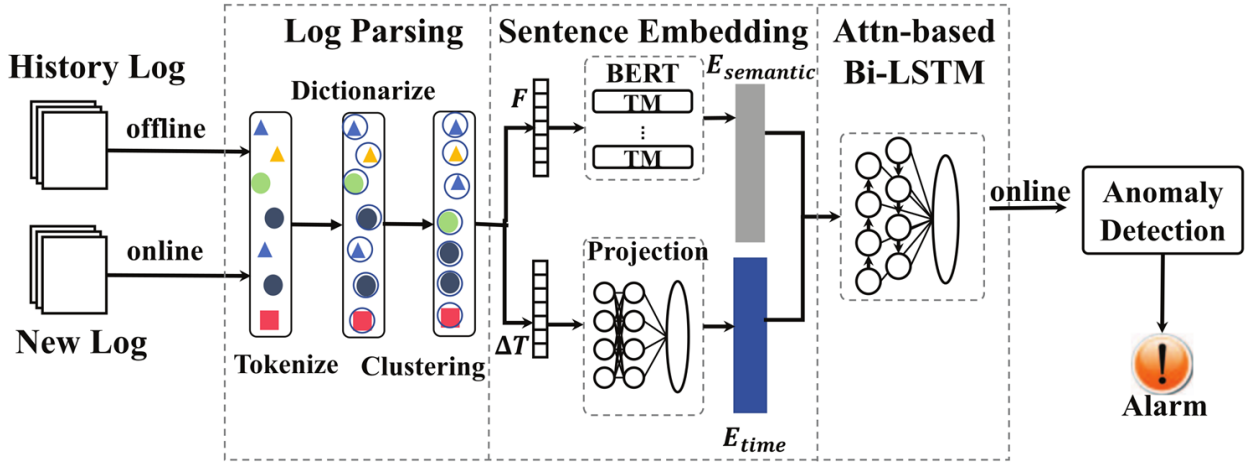


Fig. 2. The overview of SwissLog

SwissLog utilise la même technique de parsing que DeepLog, avec la plus longue sous-séquence.
L'algorithme BERT [21] (Bidirectional Encoder Representations from Transformers) est utilisé pour extraire les features contenues dans les template ajouté au temps entre les évènements, comme DeepLog à l'aide d'une suite à temps continu.
Enfin, la même méthode que LogRobust est utilisée pour détecter les anomalies.

$$\text{Precision} = \frac{TP}{TP+FP}$$

$$\text{Recall} = \frac{TP}{TP+FN}$$

$$\text{F-measure} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

TP = True Positive (same for TN)
FP = False Positive (same for FN)

A – Log parsing

Principe simple, passer d'une ligne de log à un template. Objectif : enlever toute l'information inutile.

Ex : « Dec 10 06:55:46 LabSZ sshd[24200]: Invalid user webmaster from 173.234.31.186 »

devient : « Invalid user <*> from <*> »

Nombreuses méthodes existantes depuis début 2000.
Meilleurs : Drain [1] (2017) et Brain [2] (2023)

Fonctionnement :
Drain : fixed-depth-tree

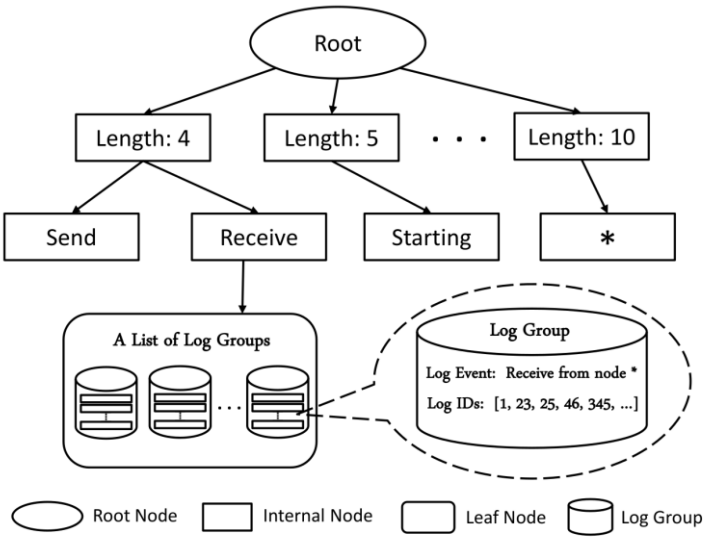


Fig. 2: Structure of Parse Tree in Drain ($depth$ = 3)

ACCURACY OF LOG PARSERS ON DIFFERENT DATASETS

Dataset	SLCT	AEL	IPLoM	LKE	LFA	LogSig	SHISO	LogCluster	LenMa	LogMine	Spell	Drain	MoLFI	Best
HDFS	0.545	**0.998**	**1***	**1***	0.885	0.850	**0.998**	0.546	**0.998**	0.851	**1***	**0.998**	**0.998**	**1**
Hadoop	0.423	0.538	**0.954**	0.670	**0.900**	0.633	0.867	0.563	0.885	0.870	0.778	**0.948**	**0.957***	**0.957**
Spark	0.685	**0.905**	**0.920**	0.634	**0.994***	0.544	**0.906**	0.799	0.884	0.576	**0.905**	**0.920**	0.418	**0.994**
Zookeeper	0.726	**0.921**	**0.962**	0.438	0.839	0.738	0.660	0.732	0.841	0.688	**0.964**	**0.967***	0.839	**0.967**
OpenStack	0.867	0.758	0.871*	0.787	0.200	0.200	0.722	0.696	0.743	0.743	0.764	0.733	0.213	0.871
BGL	0.573	0.758	**0.939**	0.128	0.854	0.227	0.711	0.835	0.69	0.723	0.787	**0.963***	**0.960**	**0.963**
HPC	0.839	**0.903***	0.824	0.574	0.817	0.354	0.325	0.788	0.830	0.784	0.654	0.887	0.824	**0.903**
Thunderb.	0.882	**0.941**	0.663	0.813	0.649	0.694	0.576	0.599	**0.943**	**0.919**	0.844	**0.955***	0.646	**0.955**
Windows	0.697	0.690	0.567	**0.990**	0.588	0.689	0.701	0.713	0.566	**0.993**	**0.989**	**0.997***	0.406	**0.997**
Linux	0.297	0.673	0.672	0.519	0.279	0.169	0.701	0.629	0.701*	0.612	0.605	0.690	0.284	0.701
Mac	0.558	0.764	0.673	0.369	0.599	0.478	0.595	0.604	0.698	0.872*	0.757	0.787	0.636	0.872
Android	0.882	0.682	0.712	**0.909**	0.616	0.548	0.585	0.798	0.880	0.504	**0.919***	**0.911**	0.788	**0.919**
HealthApp	0.331	0.568	0.822*	0.592	0.549	0.235	0.397	0.531	0.174	0.684	0.639	0.780	0.440	0.822
Apache	0.731	**1***	**1***	**1***	**1***	0.582	**1***	0.709	**1***	**1***	**1***	**1***	**1***	**1**
OpenSSH	0.521	0.538	0.802	0.426	0.501	0.373	0.619	0.426	**0.925***	0.431	0.554	0.788	0.500	**0.925**
Proxifier	0.518	0.518	0.515	0.495	0.026	**0.967***	0.517	**0.951**	0.508	0.517	0.527	0.527	0.013	**0.967**
Average	0.637	0.754	0.777	0.563	0.652	0.482	0.669	0.665	0.721	0.694	0.751	0.865*	0.605	N.A.

Résultats excellent de Drain mais Brain fait mieux.

Brain : Bidirectional parallel tree

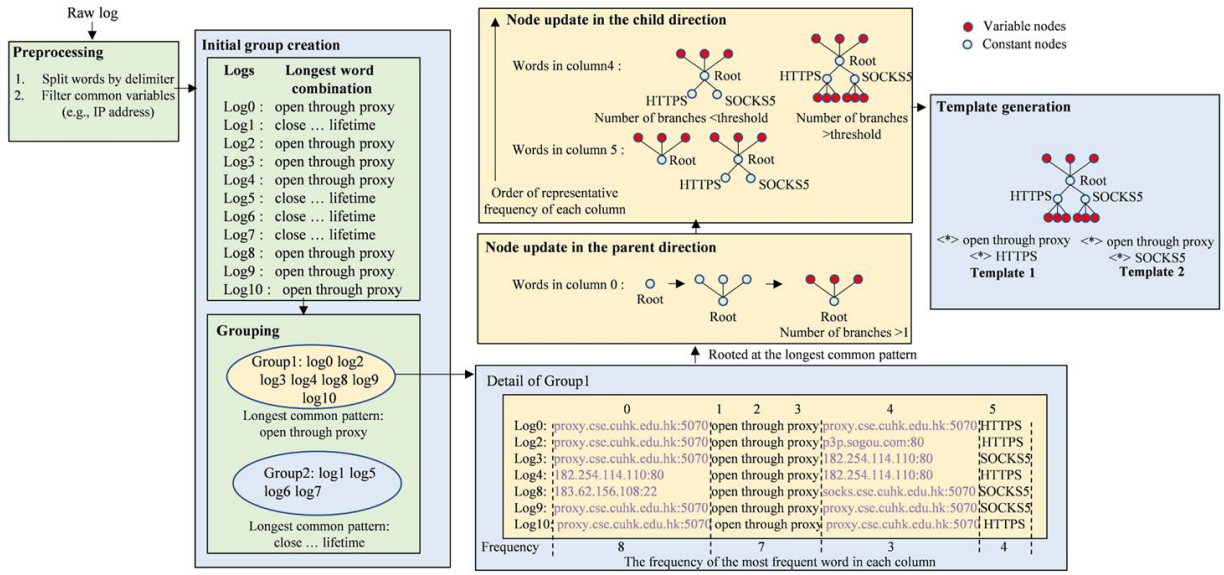
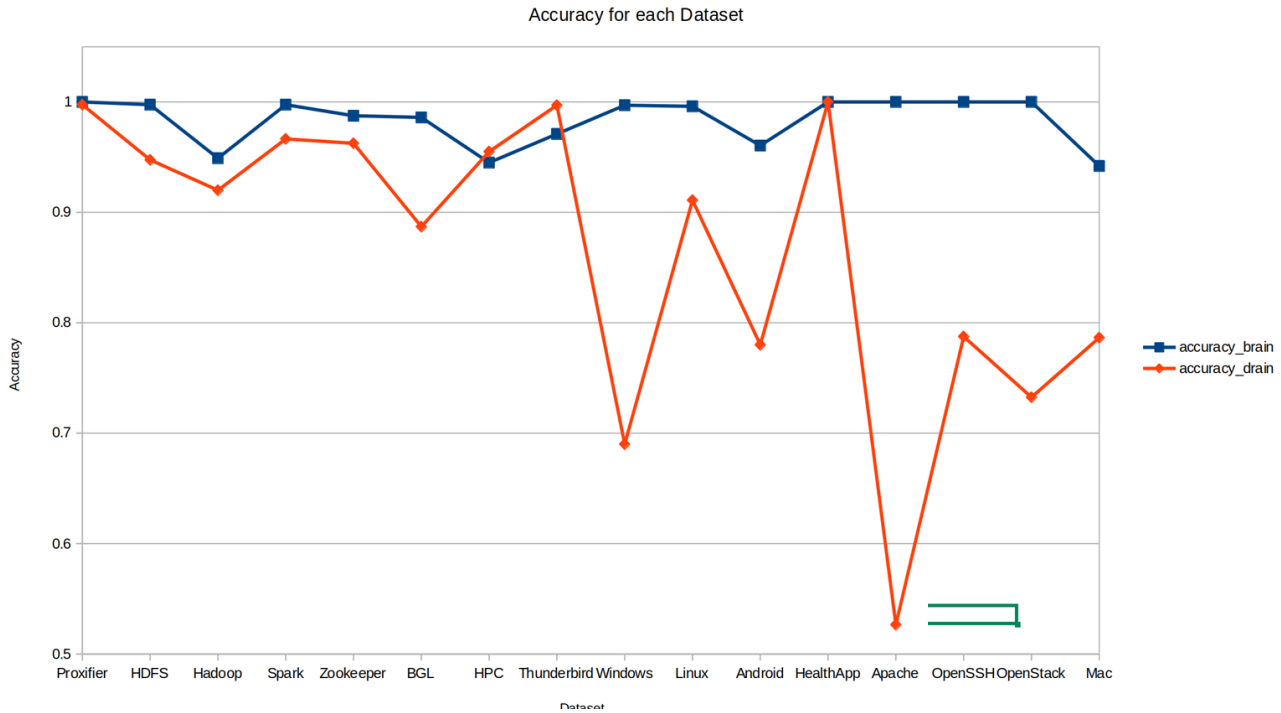


Fig. 3. The workflow of Brain parsing the log messages in Fig. 2.

Beaucoup plus efficaces que Drain.



On va se concentrer sur OpenSSH dans un premier temps. D'abord sur un sample de logs de 2.000 lignes puis sur un fichier complet de LogHub [3] de 655.000 lignes.

46s de travail pour le fichier complet.
Ex :

EventId	EventTemplate	Occurrences
E0	reverse mapping checking getaddrinfo for <*> <*> failed - POSSIBLE BREAK-IN ATTEMPT!	18909
E1	PAM <*> more authentication <*> logname= uid=0 euid=0 tty=ssh ruser= <*> <*>	37029
E2	Bad protocol version identification CONNECT <*> <*> HTTP/1.1 from <*> port <*>	1
E3	Received disconnect from <*> <*> Normal Shutdown Thank you for playing [preauth]	92
E4	Invalid <*> <*> <*> <*>	14581
E5	input_userauth_request: invalid <*> <*> <*>	14581
E6	Connection closed by <*> [preauth]	68958
E7	<*> packet length <*> [preauth]	1
E8	<*> MAC on input. [preauth]	1
E9	fatal: no hostkey alg [preauth]	6
E10	syslogin_perform_logout: logout returned an error	1

Sur un total de 47 évènements trouvés avec un F1 à 1 et un accuracy à 1. Donc tous les évènements ont été traités et ont reçu un template.

Brain se base sur Drain dans sa construction mais en allant plus loin. Lorsque le dernier regarde dans toutes les branches de l'arbre même si une feuille a déjà été observé, Brain ne regarde que les nouvelles feuilles et donc gagne en rapidité et en précision.

Tous les autres algorithmes ont une application précise et sont bon sur un type de log mais Brain reste le plus efficace [3] sur tous les types de logs observés ici.

Dataset	SLCT	AEL	IPLoM	LKE	LFA	LogSig	SHISO	LogCluster	LenMa	LogMine	Spell	Drain	MoLFI	Brain	Best	
HDFS	0.545	0.998	1	1	0.885	0.85	0.998	0.546	0.998	0.851	1	0.998	0.998	0.998	1	Spell
Hadoop	0.423	0.538	0.954	0.67	0.9	0.633	0.867	0.563	0.885	0.87	0.778	0.948	0.957	0.949	0.957	MoLFI
Spark	0.685	0.905	0.92	0.634	0.994	0.544	0.906	0.799	0.884	0.576	0.905	0.92	0.418	0.998	0.998	Brain
Zookeeper	0.726	0.921	0.962	0.438	0.839	0.738	0.66	0.732	0.841	0.688	0.964	0.967	0.839	0.988	0.988	Brain
OpenStack	0.867	0.758	0.871	0.787	0.2	0.2	0.722	0.696	0.743	0.743	0.764	0.733	0.213	1	1	Brain
BGL	0.573	0.758	0.939	0.128	0.854	0.227	0.711	0.835	0.69	0.723	0.787	0.963	0.96	0.986	0.986	Brain
HPC	0.839	0.903	0.824	0.574	0.817	0.354	0.325	0.788	0.83	0.784	0.654	0.887	0.824	0.945	0.945	Brain
Thunderb.	0.882	0.941	0.663	0.813	0.649	0.694	0.576	0.599	0.943	0.919	0.844	0.955	0.646	0.971	0.971	Brain
Windows	0.697	0.69	0.567	0.99	0.588	0.689	0.701	0.713	0.566	0.993	0.989	0.997	0.406	0.997	0.997	Brain
Linux	0.297	0.673	0.672	0.519	0.279	0.169	0.701	0.629	0.701	0.612	0.605	0.69	0.284	0.996	0.996	Brain
Mac	0.558	0.764	0.673	0.369	0.599	0.478	0.595	0.604	0.698	0.872	0.757	0.787	0.636	0.942	0.942	Brain
Android	0.882	0.682	0.712	0.909	0.616	0.548	0.585	0.798	0.88	0.504	0.919	0.911	0.788	0.961	0.961	Brain
HealthApp	0.331	0.568	0.822	0.592	0.549	0.235	0.397	0.531	0.174	0.684	0.639	0.78	0.44	1	1	Brain
Apache	0.731	1	1	1	1	0.582	1	0.709	1	1	1	1	1	1	1	Drain
OpenSSH	0.521	0.538	0.802	0.426	0.501	0.373	0.619	0.426	0.925	0.431	0.554	0.788	0.5	1	1	Brain
Proxifier	0.518	0.518	0.515	0.495	0.026	0.967	0.517	0.951	0.508	0.517	0.527	0.527	0.013	1	1	Brain
Average	0.6296875	0.7596875	0.806	0.6465	0.6435	0.5175625	0.68	0.6824375	0.766625	0.7354375	0.792875	0.8656875	0.620125	0.9830625	N.A.	13/16

Brain reste en tête sur 81 % des logs observés, on va donc garder cette méthode pour la suite.

Références :

[1] : P. He, J. Zhu, Z. Zheng and M. R. Lyu, "Drain: An Online Log Parsing Approach with Fixed Depth Tree," *2017 IEEE International Conference on Web Services (ICWS)*, Honolulu, HI, USA, 2017, pp. 33-40, doi: 10.1109/ICWS.2017.13.

[2] : S. Yu, P. He, N. Chen and Y. Wu, "Brain: Log Parsing With Bidirectional Parallel Tree," in *IEEE Transactions on Services Computing*, vol. 16, no. 5, pp. 3224-3237, Sept.-Oct. 2023, doi: 10.1109/TSC.2023.3270566.

[3] : Jieming Zhu, Shilin He, Jinyang Liu, Pinjia He, Qi Xie, Zibin Zheng, Michael R. Lyu. Tools and Benchmarks for Automated Log Parsing. ICSE, 2019.

[4] : van Ede, T., Aghakhani, H., Spahn, N., Bortolameotti, R., Cova, M., Continella, A., van Steen, M., Peter, A., Kruegel, C. & Vigna, G. (2022, May). DeepCASE: Semi-Supervised Contextual Analysis of Security Events. In 2022 Proceedings of the IEEE Symposium on Security and Privacy (S&P). IEEE.

[5] : Du, M., Li, F., Zheng, G., & Srikumar, V. (2017). Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS) (pp. 1285-1298).

[6] : T. v. Ede *et al.*, "DEEPCASE: Semi-Supervised Contextual Analysis of Security Events," *2022 IEEE Symposium on Security and Privacy (SP)*, San Francisco, CA, USA, 2022, pp. 522-539, doi: 10.1109/SP46214.2022.9833671.

[7] : Ryciak, P.; Wasielewska, K.; Janicki, A. Anomaly Detection in Log Files Using Selected Language Processing Methods. Appl. Sci. 2022, 12, 5089. https://doi.org/10.3390/app12105089

[8] : Joulin, Armand and Grave, Edouard and Bojanowski, Piotr and Mikolov, Tomas ;Bag of Tricks for Efficient Text Classification; 2017. https://arxiv.org/pdf/1607.01759

[9] : Y. Liang, Y. Zhang, H. Xiong, and R. Sahoo, “Failure prediction in ibm bluegene/l event logs,” in Seventh IEEE International Conference on Data Mining (ICDM 2007), pp. 583–588, IEEE, 2007

[10] : W. Xu, L. Huang, A. Fox, D. Patterson, and M. Jordan, “Largescale system problem detection by mining console logs,” Proceedings of SOSP’09, 2009.

[11] : J.-G. Lou, Q. Fu, S. Yang, Y. Xu, and J. Li, “Mining invariants from console logs for system problem detection.,” in USENIX Annual Technical Conference, pp. 1–14, 2010.

[12] : X. Zhang, Y. Xu, Q. Lin, B. Qiao, H. Zhang, Y. Dang, C. Xie, X. Yang, Q. Cheng, Z. Li, et al., “Robust log-based anomaly detection on unstable log data,” in Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pp. 807–817, 2019.

[13] : W. Meng, Y. Liu, Y. Zhu, S. Zhang, D. Pei, Y. Liu, Y. Chen, R. Zhang, S. Tao, P. Sun, et al., “Loganomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs.,” in IJCAI, vol. 19, pp. 4739–4745, 2019.

[14] : X. Li, P. Chen, L. Jing, Z. He, and G. Yu, “Swisslog: Robust and unified deep learning based log anomaly detection for diverse faults,” in 2020 IEEE 31st International Sym-posium on Software Reliability Engineering (ISSRE), pp. 92–103, IEEE, 2020.

[15] : K. Sharma and A. Kumar, "A Graph Database-Based Method for Network Log File Analysis," *2022 11th International Conference on System Modeling & Advancement in Research Trends (SMART)*, Moradabad, India, 2022, pp. 545-550, doi: 10.1109/SMART55829.2022.10047250.

[16] : N. Afzaliseresht, Y. Miao, S. Michalska, Q. Liu and H. Wang, "From logs to Stories: Human-Centred Data Mining for Cyber Threat Intelligence," in *IEEE Access*, vol. 8, pp. 19089-19099, 2020, doi: 10.1109/ACCESS.2020.2966760.

[17] : A. Brandao and P. Georgieva, "Log Files Analysis For Network Intrusion Detection," *2020 IEEE 10th International Conference on Intelligent Systems (IS)*, Varna, Bulgaria, 2020, pp. 328-333, doi: 10.1109/IS48319.2020.9199976.

[18] : E. W. Fulp, G. A. Fink, J. N. Haack, “Predicting Computer System Failures Using Support Vector Machines”, 2009, in https://www.usenix.org/legacy/events/wasl/tech/full_papers/fulp/fulp.pdf

[19] : Aimad Berady, Mathieu Jaume, Valérie Viet Triem Tong, Gilles Guette. PWNJUTSU: A Dataset and a Semantics-Driven Approach to Retrace Attack Campaigns. IEEE Transactions on Network and Service Management, 2022, Special Issue on Recent Advances in Network Security Management, 19 (4), pp.5252-5264. 10.1109/TNSM.2022.3183476. Hal-03694719

[20] : M. Du and F. Li, "Spell: Streaming Parsing of System Event Logs," *2016 IEEE 16th International Conference on Data Mining (ICDM)*, Barcelona, Spain, 2016, pp. 859-864, doi: 10.1109/ICDM.2016.0103.

[21] : J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “Bert: Pre-training of deep bidirectional transformers for language understanding,” arXiv preprint arXiv:1810.04805, 2018.