УДК 004.492.3
doi:10.31799/1684-8853-2026-1-36-47
EDN: LSYLOP

Научные статьи
Articles

Алгоритм автоматического построения регулярных выражений для предобработки журнальных сообщений произвольного формата в вычислительных системах

Г. А. Драчев [a,б], *соискатель, приглашенный преподаватель, руководитель проектов, orcid.org/0000-0003-1851-5507, pendal2@gmail.com*
[a] *Национальный исследовательский университет «Высшая школа экономики», Таллинская ул., 34, Москва, 123458, РФ*
[б] *Центр специальной системотехники — сервис, Варшавское ш., 71, Москва, 117556, РФ*

Введение: *предобработка журнальных сообщений необходима для структуризации журнальных сообщений и выделения полей-характеристик для последующего анализа в целях обнаружения аномалий и компьютерных атак в вычислительной системе. Наиболее приемлемый подход, который доминирует в коммерческих решениях, — построение регулярных выражений, соответствующих журнальным сообщениям, что требует трудоемкой ручной обработки.* ***Цель:*** *разработать алгоритм автоматического построения регулярных выражений журнальных сообщений в реальном масштабе времени, который можно применить к любому журнальному сообщению независимо от его формата и источника продуцирования.* ***Результаты:*** *анализ источников журнальных сообщений, способов их доставки в системы хранения и обработки, а также существующих форматов журнальных сообщений показал, что журнальные сообщения, даже в рамках одного формата, часто не стандартизованы по набору полей. Разработан алгоритм, позволяющий по тексту журнального сообщения сформировать соответствующий ему шаблон и на основе процедуры обработки шаблонов построить регулярное выражение с выделенными полями, т. е. структурировать журнальное сообщение. Помимо этого, спроектирована система хранения накопленных шаблонов журнальных сообщений и соответствующих им регулярных выражений. Для проведения экспериментальных исследований разработан программный комплекс, обеспечивающий построение регулярных выражений по текстам журнальных сообщений в автоматическом режиме. Программный комплекс апробирован на реальных вычислительных системах различных конфигураций.* ***Практическая значимость:*** *предложенный алгоритм позволяет структурировать журнальные сообщения произвольных типов и форматов. Структурированные журнальные сообщения могут быть использованы для расследования инцидентов информационной безопасности, аудита информационных систем, в качестве входных данных для анализаторов аномалий и компьютерных атак.*

Ключевые слова *— информационная безопасность, журналы вычислительной системы, журнальные сообщения, предобработка данных, структуризация журнальных сообщений, регулярное выражение.*

Для цитирования: Драчев Г. А. Алгоритм автоматического построения регулярных выражений для предобработки журнальных сообщений произвольного формата в вычислительных системах. *Информационно-управляющие системы*, 2026, № 1, с. 36–47. doi:10.31799/1684-8853-2026-1-36-47, EDN: LSYLOP

For citation: Drachev G. A. Algorithm for automatic construction of regular expressions for preprocessing arbitrary-format log messages in computing systems. *Informatsionno-upravliaiushchie sistemy* [Information and Control Systems], 2026, no. 1, pp. 36–47 (In Russian). doi:10.31799/1684-8853-2026-1-36-47, EDN: LSYLOP

Введение

Вычислительные системы в ходе своей жизнедеятельности порождают огромный массив журнальных сообщений (ЖС) [1]. Источником ЖС в контуре вычислительной системы (ВС) может являться любой ее программный или аппаратный компонент. Транспортировка и хранение ЖС осуществляются различными протоколами ВС (SYSLOG, https://www.rfc-editor.org/rfc/rfc5424; SNMP, https://www.rfc-editor.org/rfc/rfc1902; журналируемыми файловыми системами и др. [2]). В работе [2] представлены примеры источников журнальных сообщений ВС.

Журнальное сообщение — это запись произвольного формата, которая отражает текущее изменение состояния одной из частей ВС. Каждая запись ЖС состоит из полей, несущих основную информационную нагрузку (временную метку, имя/идентификатор процесса, имя/идентификатор пользователя и др.), а также сопроводительных текстовых данных. В процессе журналирования ВС накапливается большое количество сообщений, которые позволяют анализировать поведение ВС и диагностировать возникающие ошибки, помогают в сопровождении ВС, в расследовании инцидентов информационной безопасности и компьютерных атак [3].

Журнальные сообщения по мере накопления записываются в файлы для их хранения (журналы). Для анализа журналов используются различные методы, например, машинного обучения [3, 4], анализа больших данных [5] и др. [6–8]. Абсолютное большинство из них не применимо для обработки ЖС в режиме реального

времени и (или) имеет ограничения по обработке форматов ЖС.

Единого стандарта для формирования ЖС нет, — программные и аппаратные источники формируют ЖС в различных форматах [1]. ЖС могут отличаться по набору, составу, количеству и смысловой нагрузке полей. Эти факторы не позволяют напрямую использовать рассмотренные методы обработки ЖС. Поэтому информацию из журнальных сообщений перед этапом анализа необходимо предварительно обработать — выделить поля, характеризующие ЖС, т. е. структурировать.

В научных работах предлагается множество подходов к решению задачи структуризации ЖС. Например, ручное формирование регулярных выражений (РВ) [2, 6] или правил разбора ЖС [7–10], анализ дерева фиксированной глубины [11], алгоритм кластеризации для создания шаблонов ЖС [12], метод построения FP-деревьев [13], алгоритм предобработки, основанный на выделении и анализе только одной «ключевой» части (характеристики) ЖС [5], статистические методы [14–17], алгоритмы машинного обучения [18–20], в частности большие языковые модели [21–24].

Существующим подходам структуризации ЖС присущ ряд функциональных недостатков:

— математические [12, 13] и статистические алгоритмы [14–17] не позволяют осуществлять обработку в режиме реального времени;

— алгоритмы машинного обучения не позволяют осуществлять качественную обработку ЖС произвольных форматов [18–20] либо не предназначены для обработки в режиме реального времени [21–24];

— алгоритмы, ориентированные на реализацию в режиме реального времени, ограничены

■ ***Таблица 1.*** Методы структуризации журнальных сообщений, применяемые в актуальных программных решениях
■ ***Table 1.*** Log structuring methods in modern software

Продукт	Основной метод структуризации	Недостатки метода структуризации
Elastic Stack (https://www.elastic.co/elastic-stack)	Регулярные выражения (шаблоны для языковой модели Grok)	Высокая сложность написания и отладки сложных шаблонов Регулярные выражения требуют точного соответствия формату. Любое отклонение приводит к ошибке структуризации ЖС Высокая нагрузка на процессор
Splunk (https://www.splunk.com/)	Конфигурирование правил разбора ЖС (props.conf) и поисковый язык (SPL)	Автоматическое извлечение полей производится с большим количеством лишних данных Ограничения по количеству извлекаемых полей Сложный алгоритм структуризации замедляет обработку в режиме реального времени
Graylog (https://graylog.org/)	Конфигурирование правил разбора ЖС	Неэффективен для произвольных форматов ЖС Затруднение отладки сложных цепочек правил Производительность падает при большом количестве правил
Grafana Loki (https://grafana.com/oss/loki/)	Описание меток (для выделения полей) и отложенная структуризация	Эффективность системы падает при увеличении количества и сложности описания меток Одновременная обработка ЖС произвольных форматов очень медленная
Vector (https://vector.dev/)	Декларативный язык (VRL) с использованием встроенных функций	Требуется изучение синтаксиса VRL для сложных преобразований Сложная отладка конфигурации
Fluentd / Bit (https://fluentbit.io/)	Плагины (на основе Regex, JSON и др.)	Производительность и возможности сильно зависят от выбранного плагина Использование множества плагинов усложняет конфигурацию и может привести к ошибкам структуризации, вплоть до нештатного завершения программы
Datadog (https://www.datadoghq.com/)	Шаблоны для языковой модели Grok	Интерфейс программы ограничивает возможности задания шаблонов Сложные нестандартные форматы все равно требуют ручного написания шаблонов
DeepLog (https://github.com/Thijs-vanede/DeepLog)	Предустановленные обработчики под журналы программного и аппаратного обеспечения от конкретных производителей	Плохая приспособленность для обработки уникальных форматов данных Зависимость от производителей компонентов ВС в обновлении и поддержке правил структуризации для нового оборудования/программного обеспечения

либо форматом ЖС [2, 7, 8, 10], либо количеством выделяемых характеристик [4, 6, 9, 11] и не позволяют структурировать редко встречающиеся ЖС.

В современных коммерческих продуктах доминирует подход к структуризации ЖС с помощью РВ или правил структуризации, составленных «вручную» (табл. 1).

Такой подход, с одной стороны, не ограничен одной выделяемой характеристикой ЖС, а с другой стороны, РВ можно подобрать для любого отдельного формата ЖС. Однако ручное формирование РВ не позволяет обрабатывать произвольные ЖС из-за разнообразия их форматов и содержания (из-за отсутствия единой стандартизации невозможно учесть все возможные варианты).

Для замены «ручного» формирования регулярного выражения предлагается разработать алгоритм автоматического построения регулярного выражения для произвольного журнального сообщения. В качестве входных данных используется журнальное сообщение произвольного формата от произвольного источника, в качестве выхода — сформированное регулярное выражение, которое соответствует журнальному сообщению, поданному на вход. Регулярное выражение должно содержать группы для структуризации журнального сообщения.

Формирование шаблонов журнальных сообщений

Большинство программных или аппаратных компонентов ВС генерирует массив ЖС различных форматов. Некоторые ЖС, например по протоколу SYSLOG, стандартизируют доставляемые сообщения при помощи добавления заголовка в начало сообщения, но не вносят изменений в неструктурированное содержание самих сообщений. Другие ЖС, например по протоколу SNMP, накладывают ограничения на формат сообщений. Сообщения по протоколу SNMP представляют собой последовательность OID [3] ключей и их значений через разделитель «;». Однако деревья OID ключей не стандартизированы и могут отличаться даже в рамках аналогичных источников ЖС (https://www.cisco.com/c/en/us/support/docs/smb/switches/Cisco-Business-Switching/kmgmt3636-snmpv3-common-oids-cbs350.html и https://community.cisco.com/t5/networking-knowledge-base/oid-list/ta-p/3117547). Отдельные журналы, например, генерируемые аппаратно-программным модулем доверенной загрузки (АПМДЗ, https://www.udcs.ru/catalog/sredstva-zashchity-informatsii/zashchita-ot-nesanktsionirovannogo-dostupa/apparatno-programmnyy-modul-doverennoy-zagruzki-apmdz-sobol/) или системой PARSEC (https://wiki.astralinux.ru/pages/viewpage.action?pageId = 67112737), имеют свои уникальные форматы ЖС и не используют возможности операционной системы для ведения журналов.

Однако вне зависимости от формата каждое ЖС представляет собой последовательность символов, которая запрограммирована заранее, исключая параметры, которые передаются в строку при ее продуцировании [8]. Вместо последовательностей символов можно рассматривать каждое ЖС как последовательность слов. Слово — это цепочка символов, ограниченная символами-разделителями.

Эмпирическим путем были выделены множества символов-разделителей (https://www.pcre.org/pcre2.txt) между словами ЖС:

[:space:] — множество управляющих символов, обозначающих разрыв между словами;

[:blank:] — множество пробельных символов;

[:punct:] — множество знаков пунктуации.

Цепочка символов, заключенная в кавычки, одинарные или двойные, учитывается как одно слово целиком, вне зависимости от наличия символов-разделителей внутри цепочки.

Назовем совпадающую по смыслу последовательность слов и символов-разделителей шаблоном ЖС. Формирование шаблона ЖС — процесс разбиения записи ЖС на слова и символы-разделители. Шаблон уже можно использовать как РВ, однако поиск по шаблону не «среагирует» на следующее ЖС того же типа, если в ЖС поменяется хотя бы одно слово. Отдельные слова у разных экземпляров одного типа ЖС могут отличаться (например, “user1” и “user2” (рис. 1)). Эти слова могут содержать важную информацию, характеризующую событие, описанное в ЖС:

— временную метку, когда было зафиксировано событие;

— субъект, который совершил или спровоцировал журналируемое действие;

— объект, над которым было совершено действие или изменение его состояния;

— результат журналируемого воздействия и др.

На рис. 1 представлен пример двух шаблонов ЖС одного типа, доставленных по протоколу SYSLOG от PAM модуля (https://uchet-jkh.ru/i/nastroika-pam-v-astra-linux/) (подключаемого модуля аутентификации) Unix-подобных ОС. ЖС содержат информацию об успешном открытии сессии по протоколу SSH в разное время для пользователей user1 и user2. В шаблонах выделены слова (поля), характеризующие событие ЖС.

Для определения полей-характеристик ЖС необходимо преобразовать его шаблон к общему виду, который будет подходить для всех ЖС того же типа.

<86>	Aug 31 15:20:21	dr-1 sshd[26272]:pam_unix(sshd:session): session opened for	user 1	user by (uid= 0)
<86>	Aug 31 18:36:51	dr-1 sshd[26272]:pam_unix(sshd:session): session opened for	user 2	user by (uid= 1)

общая часть поля-характеристики

■ ***Рис. 1.*** Два шаблона журнальных сообщений
■ ***Fig. 1.*** Two log message templates

Анализ и преобразование шаблонов журнальных сообщений

Для определения принадлежности шаблонов ЖС к одному типу и их преобразования к шаблону общего вида предложен механизм (анализатор шаблонов), который сравнивает новый шаблон с теми, которые были получены ранее. Для операции сравнения выбираются шаблоны совпадающей длины (определяется в словах) и подпоследовательности символов-разделителей. Результат работы анализатора шаблонов — это решение о внесении в базу шаблонов нового шаблона или изменении (выделении полей), при необходимости, одного из старых (т. е. приведение к общему виду). Если для нового шаблона нет совпадения в базе старых шаблонов, то он вносится как новая запись. Если находится шаблон (1) (рис. 2), который отличается от нового (2) не больше чем на половину слов от длины шаблона, то оба шаблона описывают один и тот же тип ЖС и должны быть преобразованы в шаблон общего вида (3). Если можно выделить набор слов, не совпадающий в сравниваемых шаблонах одного типа ЖС, то в шаблоне общего вида (3) слова такого набора будем считать изменяющимися полями-характеристиками (обозначим каждый из них символом «*»). Преобразование шаблонов (1) и (2) в шаблон общего вида (3) с выделением слов-характеристик представлен на рис. 2.

Поле Wed Feb 1 06:53:59 2023, представляющее собой временную метку ЖС, заключено в одинарные кавычки и поэтому воспринимается одним словом (см. рис. 2). Однако чаще всего (например, в SYSLOG, АПМДЗ и др.) временная метка не закрывается кавычками, что приводит к разбиению временной метки на отдельные поля. Чтобы сохранить целостность временной метки, предлагается использовать готовое решение: определять временную метку в ЖС при помощи функции из библиотеки timeGrinder (https://pkg.go.dev/github.com/gravwell/gravwell/v3/timegrinder#Extract) для ее обработки как целостного поля-характеристики.

Шаблон общего вида, полученный в результате работы анализатора шаблонов, описывает ЖС одного типа и позволяет определить поля-характеристики этих ЖС. Шаблон общего вида можно преобразовать в РВ при помощи замены «*» на «(.*)» (в РВ — любая последовательность символов), т. е. трансформировать конструкцию в группу (https://www.pcre.org/pcre2.txt).

Алгоритм построения регулярных выражений

Сконструируем формальный алгоритм построения РВ на основе ЖС произвольных форматов.

Журнальное сообщение 1:													
[f]'Wed Feb 1 06:53:59 2023' '/bin/atsrv' <21588,12087,0,0,0> [s] open("/usr/argusids/lib64/libtcryptf.so",NO_PERMS | O_LARGEFILE) = 0													
Журнальное сообщение 2:													
[f]'Wed Feb 1 06:54:04 2023' '/bin/argsh0' <21599,12087,0,0,0> [s] open("/bin/atsrv",NO_PERMS | O_LARGEFILE) = 0													
Шаблон (1) (для сообщения 1):													
f	Wed Feb 1 06:53:59 2023	/bin/atsrv	21588	12087	0	0	0	s	open	/usr/argusids/lib64/libtcryptf.so	NO_PERMS	O_LARGEFILE	0
Шаблон (2) (для сообщения 2):													
f	Wed Feb 1 06:54:04 2023	/bin/argsh0	21599	12087	0	0	0	s	open	/bin/atsrv	NO_PERMS	O_LARGEFILE	0
Шаблон общего вида (3):													
f	*	*	*	12087	0	0	0	s	open	*	NO_PERMS	O_LARGEFILE	0

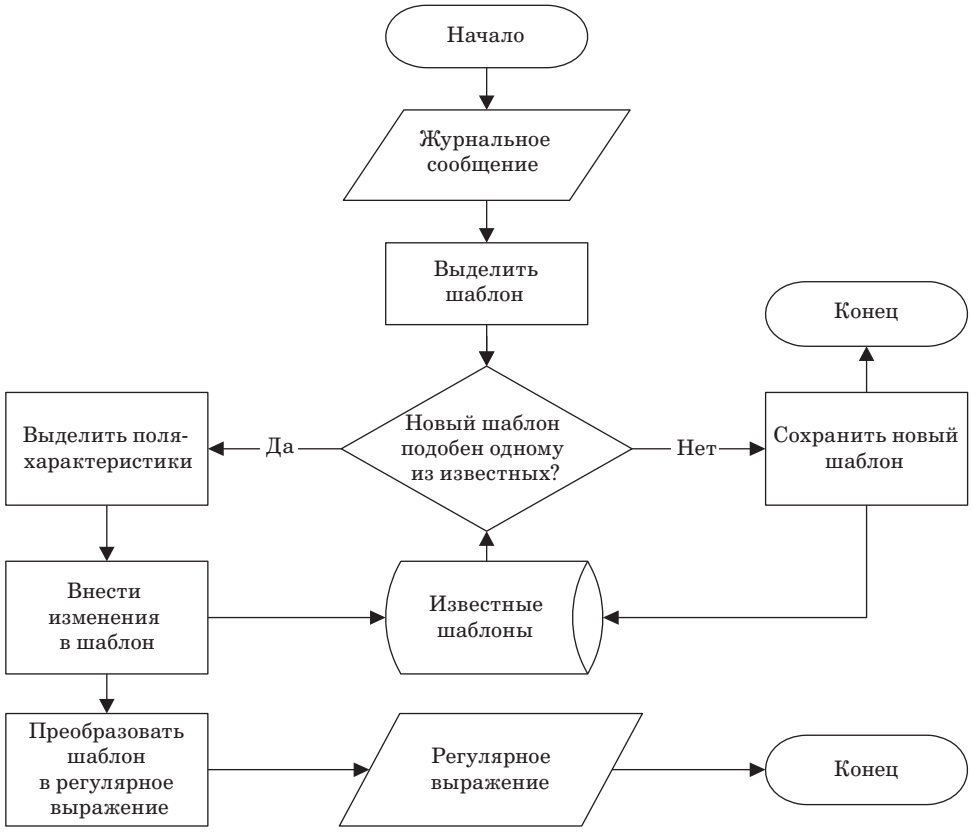
■ ***Рис. 2.*** Преобразование шаблона (1) в (3) по результатам сравнения с шаблоном (2)
■ ***Fig. 2.*** Conversion of template (1) to (3) after matching against template (2)

Шаг 1. По одному из протоколов $p \in P$, где P — протоколы транспортировки сообщений, на вход поступает информация $m_n^{s,p} \in M$, где M — информация в виде цепочки символов $\left\{m_1^{s,p}, ..., m_N^{s,p}\right\}$, N — максимальное возможное количество вариантов информации от известных в рамках конфигурации ВС источников $s \in S$, S — источники $\{s_1, ..., s_A\}$, где A — количество источников информации в ВС.

Шаг 2. Полученная цепочка $m_n^{s,p}$ разбивается на последовательность слов $\left(\left(w_q\right)_1^Q\right)^{m_n^{s,p}}$, т. е. формируется шаблон, где Q — количество слов в цепочке. Разбиение осуществляется путем поиска разделителей $t \in T$, $T = \{t_1, ..., t_Z\}$, $1 \le z \le Z$, где Z — количество вариантов символов-разделителей. Обозначим подпоследовательность символов-разделителей для шаблона $\left(\left(w_q\right)_1^Q\right)^{m_n^{s,p}}$ как $T_Q^{m_n^{s,p}}$.

Шаг 3. Новый шаблон $\left(\left(w_q\right)_1^Q\right)^{m_n^{s,p}}$ сравнивается с уже известными шаблонами из множества шаблонов общего вида $W = \left\{\left(\left(w_y\right)_1^Y\right)^{m_1^{s,p}}, ..., \left(\left(w_y\right)_1^Y\right)^{m_N^{s,p}}\right\}$, у которых такая же длина $Q = Y$ и такая же подпоследовательность символов-разделителей $T_Q^{m_n^{s,p}} = T_Y^m$.

Шаг 4. Если для шаблона $\left(\left(w_q\right)_1^Q\right)^{m_n^{s,p}}$ найдется такой $\left(\left(w_y\right)_1^Y\right)^m \in W$, что хотя бы половина их слов совпадает, т. е. $\left(\left(w_q\right)_1^Q\right)^{m_n^{s,p}} \cap \left(\left(w_y\right)_1^Y\right)^m \ge \left(w_q\right)_1^{\left[\frac{Q}{2}\right]}$, то считаем, что $\left(\left(w_q\right)_1^Q\right)^{m_n^{s,p}}$ и $\left(\left(w_y\right)_1^Y\right)^m$ подобны (соответствуют одному и тому же типу ЖС, см. рис. 2). Тогда элементы w, которые составляют разницу между $\left(\left(w_q\right)_1^Q\right)^{m_n^{s,p}}$ и $\left(\left(w_y\right)_1^Y\right)^m$, определяем как поля-характеристики $\{f_1, ..., f_J\}$, где J — максимально возможное количество полей для $\left(\left(w_q\right)_1^Q\right)^m$. Перейти к шагу 6.



■ ***Рис. 3.*** Алгоритм построения регулярных выражений

■ ***Fig. 3.*** Flowchart of the regular expression construction algorithm

Шаг 5. Если для шаблона $\left(\left(w_q\right)_1^Q\right)^{m_n^{s,p}}$ не найдется такой $\left(\left(w_y\right)_1^Y\right)^m \in W$, что хотя бы половина их слов совпадает, т. е. $\left(\left(w_q\right)_1^Q\right)^{m_n^{s,p}} \cap \left(\left(w_y\right)_1^Y\right)^m \geq \geq \left(w_q\right)_1^{\left\lceil \frac{Q}{2} \right\rceil}$, то считаем, что $\left(\left(w_q\right)_1^Q\right)^{m_n^{s,p}}$ – новый уникальный шаблон (новый тип ЖС). Завершить итерацию алгоритма.

Шаг 6. Для $\left(\left(w_q\right)_1^Q\right)^{m_n^{s,p}}$ выполняется преобразование $\mathrm{Re}\, g\left(\left(\left(w_q\right)_1^Q\right)^{m_n^{s,p}}, T_Q^{m_n^{s,p}}\right) = r_h$, $h \in [1; H]$, $r_h \in R$, где H – количество полученных шаблонов; R – множество регулярных выражений [2]. В рамках преобразования слова, выделенные на шаге 4 как поля, заменяются на «.*» и выделяются в группы РВ r_h. Группы считаются, начиная с 1-й, группа 0 – все сообщение целиком. Завершить итерацию алгоритма.

Представленный алгоритм можно изобразить в виде блок-схемы (рис. 3).

Оценка временной сложности алгоритма построения РВ: $O(q) + O(k) << \infty$, где q – количество слов в шаблоне ЖС, а k – количество старых шаблонов, которые удовлетворяют условиям $Q = Y$ и $T_Q^{m_n^{s,p}} = T_Y^m$.

Реализация системы построения регулярных выражений

Для эффективной работы алгоритма построения РВ необходимо реализовать систему хранения накопленных шаблонов ЖС и соответствующих им РВ. Для решения этой проблемы была спроектирована база данных (БД) SQlite3 (https://www.sqlite.org/). БД представляется в виде двух таблиц:

– templates – таблица для хранения шаблонов:

- id – идентификатор записи шаблона;
- template – строка для хранения шаблона ЖС;

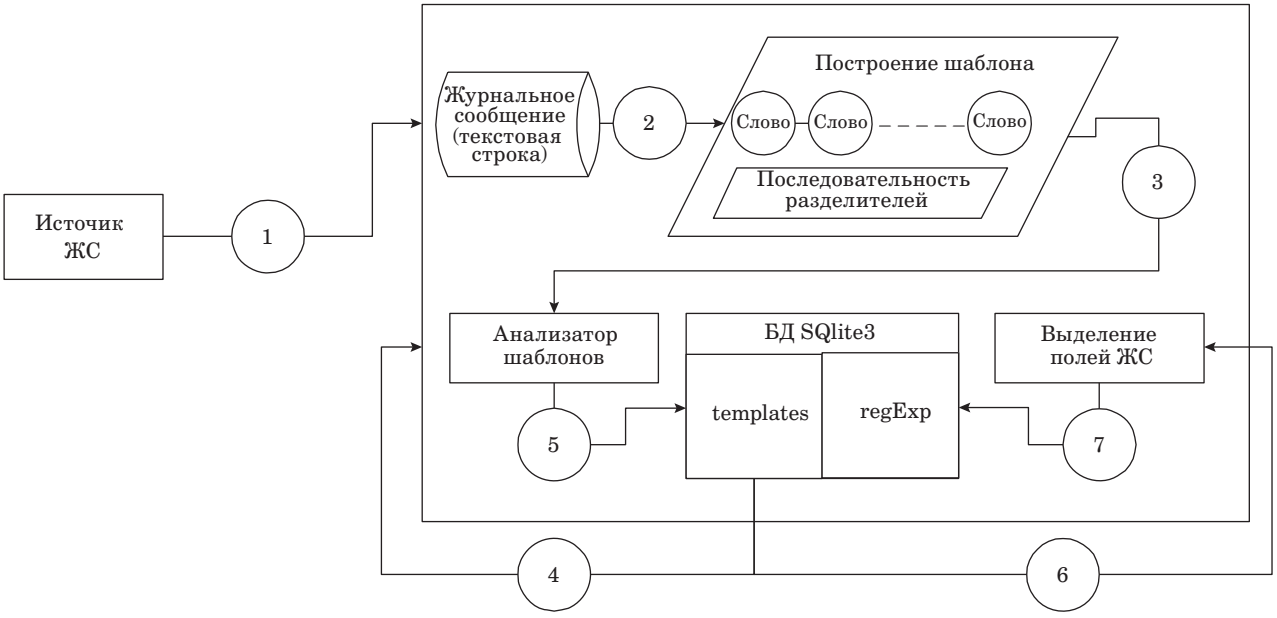
– regExp – таблица для хранения РВ:

- id – целочисленный идентификатор записи РВ;
- idt – целочисленный идентификатор записи шаблона, которому соответствует запись РВ;
- regular – строка для хранения РВ.

Записи шаблонов и соответствующих им РВ в таблицах templates и regExp связаны через идентификаторы биективно. Если в процессе работы алгоритма построения РВ какой-либо шаблон будет скорректирован в соответствии с шагом 3 алгоритма, то соответствующая ему запись РВ в БД будет также изменена.

Обобщенная схема программной реализации построения РВ ЖС представлена на рис. 4.

1. Журнальное сообщение создается источником и отправляется по одному из протоколов.



■ ***Рис. 4.*** Этапы получения и преобразования записей ЖС в РВ

■ ***Fig. 4.*** Steps involved in the collection and conversion of log entries to regular expressions

2. На основании текстовой строки ЖС создается его шаблон.

3. Шаблон передается в анализатор.

4. Из таблицы templates запрашиваются уже существующие шаблоны такой же длины и с той же последовательностью символов-разделителей.

5. Шаблон ЖС сравнивается с полученными шаблонами из БД. На основании результатов сравнения анализатором принимается решение о приведении шаблонов к общему виду, либо о внесении нового шаблона, либо об отсутствии необходимости в этих действиях.

6. В шаблоне общего вида выделяются поля (для структуризации ЖС).

7. Шаблон общего вида с выделенными полями, характеризующими ЖС (регулярное выражение), вносится в таблицу regExp как новое или измененное (в соответствии с п. 6) РВ.

Обобщенная схема реализована в форме программного комплекса, позволяющего преобразовывать ЖС в РВ.

Экспериментальные исследования

Апробация алгоритма построения РВ производилась как на физических, так и на виртуальных устройствах, которые были включены в реальные ВС различных конфигураций

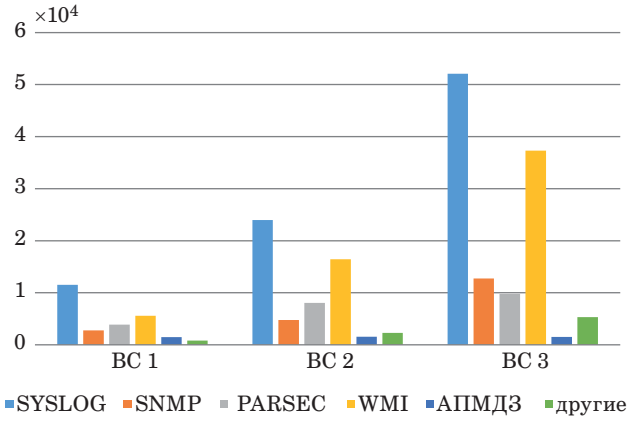
ВС 1. Локальная сеть с выходом в сеть Интернет, включающая: 15 компьютеров под управлением ОС Astra linux 1.5, семь компьютеров под управлением ОС Astra linux 1.6, 10 компьютеров под управлением ОС Astra linux 8.1, 12 компьютеров под управлением ОС Windows 10, два компьютера под управлением ОС Windows Server 2012, три устройства Cisco 2960, два устройства HUAWEI WiFi AX2, три устройства Dionis NX, два устройства Canon MF461DW.

ВС 2. Локальная сеть без выхода в сеть Интернет, включающая: пять компьютеров под управлением ОС Alt linux, 18 компьютеров под управлением ОС Astra linux 1.6, 27 компьютеров под управлением ОС Astra linux 8.1, 12 компьютеров под управлением ОС Windows 10, 14 компьютеров под управлением ОС Windows 11, 12 компьютеров под управлением ОС Windows 8.1, один компьютер под управлением ОС CentOS linux, пять устройств Cisco 2951, три устройства Juniper MX Series, одно устройство MikroTik CRS504-4XQ-IN, одно устройство RUSTELETEH RTT-M300.

ВС 3. Локальная сеть с выходом в сеть Интернет, включающая: 35 компьютеров под управлением ОС Astra linux 1.6, 27 компьютеров под управлением ОС Astra linux 8.1, 54 компьютера под управлением ОС Windows 10, 43 компьютера под управлением ОС Windows 11, 41 компьютер под управлением ОС CentOS linux, пять компьютеров под управлением ОС Ubuntu linux, два устройства Canon MF461DW, 23 устройства D-link DSA-2208X, восемь устройств MikroTik CRS504-4XQ-IN, два устройства «Поток КМ-122», 15 межсетевых экранов ССПТ2.

Вычислительные системы, построенные для проведения испытаний, продуцировали ЖС разных форматов (рис. 5). Оценка скорости появления новых ЖС показала неравномерные результаты, зависящие от различных факторов: количества аппаратных и программных компонент, составляющих ВС, их загруженности, установленных конфигураций журналирования, количества ошибок в ВС и других внешних факторов (например, обработки запросов от клиентов на интернет-странице). В зависимости от изложенных аспектов скорость наполнения журналов сообщениями может варьироваться. Испытания алгоритма построения РВ производились в течение 168 ч непрерывного эксперимента для каждой ВС во время их эксплуатации. За этот период минимальный объем получаемых сообщений был зафиксирован на уровне 2352 сообщений в секунду, а максимальный — 739 681 сообщения в секунду. В среднем же представленные ВС продуцировали 60 000–70 000 сообщений в секунду.

Можно заметить (см. рис. 5), что основной объем ЖС во всех ВС сгенерирован в формате SYSLOG. Это объясняется тем, что SYSLOG — базовая система журналирования для всех UNIX подобных операционных систем. Благодаря этому программное обеспечение, написанное для linux-систем, чаще всего использует возможности, предоставляемые операционной системой (SYSLOG) для ведения журналов. Кроме того,



■ ***Рис. 5.*** Распределение журнальных сообщений по протоколам доставки в наблюдаемых системах, продуцируемых в среднем за секунду

■ ***Fig. 5.*** Log message distribution by transport protocols across monitored systems (average per-second generation rate)

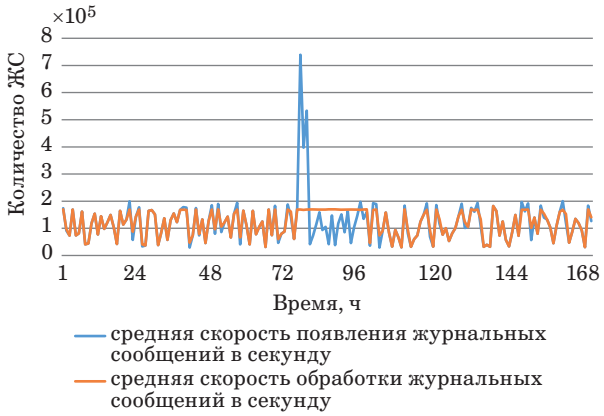
в телекоммуникационном оборудовании чаще всего используются UNIX подобные операционные системы, что также сказывается на количестве ЖС, генерируемых ВС в этом формате. Протокол SNMP используется намного реже, чем SYSLOG (см. рис. 5). Несмотря на то, что протокол SNMP можно настроить как для ОС linux, так и ОС Windows, чаще всего он используется только в телекоммуникационном оборудовании, где настроен сразу на базовом уровне, ввиду удобства использования возможности активных GET-запросов для опроса состояния сетевого оборудования. Система журналирования PARSEC также не отличается большим количеством продуцируемых экземпляров сообщений, так как является частью одноименной системы разграничения прав доступа и генерирует сообщения, отражающие только важные события, связанные с ней. Протокол WMI [25] является внутренним решением для ОС Windows. Количество сообщений, создаваемых в этом формате в среднем, пропорционально количеству устройств, использующих ОС Windows в ВС. Количество сообщений АПМДЗ в целом остается на одном уровне для всех ВС, так как зависит от количества используемых физических плат (которых было в недостаточном количестве — всего по две платы в каждой ВС). Другие ЖС в ВС продуцировались отдельными экземплярами программного обеспечения, которые не использовали штатные средства операционных систем для ведения журналов.

Первоначально программная реализация алгоритма построения РВ тестировалась на компьютере на базе процессора «Эльбрус 8СВ» (архитектура e2k, восемь физических ядер (http://mcst.ru/Elbrus-8CB)), под управлением ОС Astra linux 8.1. Данный компьютер присутствовал во всех тестируемых ВС. Обработка ЖС осуществлялась со скоростью 3700–4000 сообщений в секунду. Этого было недостаточно для обработки всего объема ЖС в режиме реального времени. Поэтому был применен метод вертикального масштабирования. Блок обработки информации в алгоритме построения РВ был распараллелен на этапе выделения шаблонов на шесть потоков. Одно ядро процессора отводилось под сведение результатов выделения шаблонов и сравнение с другими шаблонами из БД. Еще одно ядро было зарезервировано для операционной системы, чтобы не потерять управление в случае чрезмерной нагрузки. В таком режиме удалось достичь ощутимой прибавки в производительности: скорость обработки сообщений возросла до 19 000–20 000 в секунду. Полученной производительности оказалось достаточно для обработки сообщений ВС 1 в реальном времени. Однако для обработки журналов ВС 2 и ВС 3 пришлось применить метод горизонтального масштабирования (разделение обработки ЖС между тремя компьютерами для ВС 2 и шестью компьютерами для ВС 3 на базе процессора «Эльбрус 8СВ»).

Были проведены дополнительные исследования на компьютере на базе процессора Intel Xeon Platinum 8362 (архитектура x86, 32 физических ядра, 64 логических ядра (https://www.intel.com/content/www/us/en/products/sku/217216/intel-xeon-platinum-8362-processor-48m-cache-2-80-ghz/specifications.html)), под управлением ОС Astra linux 1.6. На тех же шести потоках получилось достичь скорости 25 000–27 000 сообщений в секунду. При увеличении количества потоков выделения шаблонов до 20 штук прирост производительности наблюдался близким к линейному и составил 83 000–85 000 сообщений в секунду, однако затем, с увеличением потоков обработки, прирост производительности начал замедляться из-за ограничения производительности процессора на одно ядро в задаче сведения выделенных шаблонов и их сравнения. Если производительность алгоритма не удовлетворяет требованиям реального времени, то необходимо использовать либо процессор с большей производительностью на ядро (при том же количестве ядер), либо метод горизонтального масштабирования.

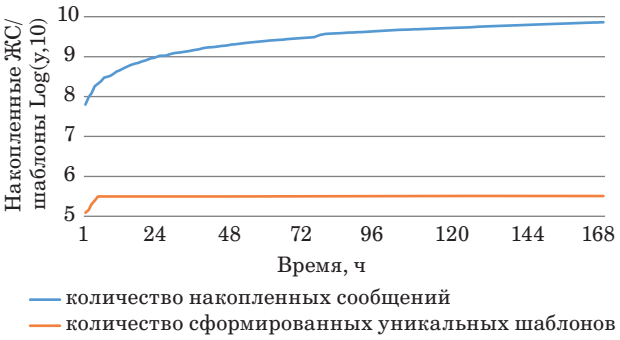
В целом все системы (ВС 1, ВС 2 и ВС 3) похожи в плане форматов ЖС и отличаются в количестве продуцируемых ЖС. Для ВС 3, генерирующей наибольший объем ЖС, из тестируемых ВС были проведены дополнительные исследования по обработке ЖС алгоритмом. ЖС обрабатывались на двух компьютерах под управлением ОС Astra linux 1.6, построенных на процессорах Intel Xeon Platinum 8362.

На графике (рис. 6) отображено сравнение скоростей разрастания журналов (среднее количество ЖС, генерируемых за секунду в течение



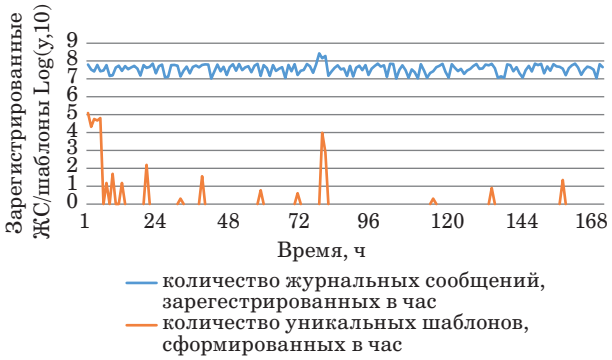
■ ***Рис. 6.*** Результаты апробации алгоритма генерации регулярных выражений в контуре ВС 3

■ ***Fig. 6.*** Validation results of the regex generation algorithm in computing system 3's environment



■ ***Рис. 7.*** Влияние роста журналов на накопление новых шаблонов в ВС 3

■ ***Fig. 7.*** Correlation between log size growth and pattern discovery rate in computing system 3



■ ***Рис. 8.*** Влияние роста журналов (журнальных сообщений) на количество сформированных уникальных шаблонов в час в ВС 3

■ ***Fig. 8.*** Effect of increasing log volume on hourly unique pattern generation rate in computing system 3

■ ***Таблица 2.*** Примеры структуризации ЖС

■ ***Table 2.*** Sample structured log message representations

Протокол	Журнальное сообщение	Регулярное выражение	Выделенные поля
SYSLOG	Dec 3 19:17:01 a161 CRON[25552]: pam_unix(cron:session): session opened for user dr1 by (uid = 1)	(.*) a161 CRON\[(.*)\]\: pam_unix\(cron\:session\)\: session opened for user (.*) by \(uid = (.*)\)	Dec 3 19:17:01 25552 dr1 1
PARSEC	[f] 'Tue Feb 18 06:53:59 2025' '/bin/atsrv' <21588,12096,0,0,0> [s] open("/usr/argusids/lib64/libtransp.so",NO_PERMS | O_LARGEFILE) = 0	\[f\] \'(.*)\' \'(.*)\' \<(.*)\,(.*)\,0\,0\,0\> \[s\] open\(\"(.*)\"\,NO_PERMS | O_LARGEFILE\) = 0	Tue Feb 18 06:53:59 2025 /bin/atsrv 21588 12096 /usr/argusids/lib64/libtransp.so
АПМДЗ	54;Log entry:p_support;User:218f0b8a-834d-0e74-b42c-aca4636cda97;Type:Событие НСД;Code:LOG_CRC_MISMATCH_FILE;File:$MountPoint/1/home/user/apmdz/log/ahsm_log;Severity:LOG_SEVERITY_ALERT;Log DateTime: Mon Feb 17 10:06:12 2025	(.*);Log entry:(.*);User:(.*);Type:Событие НСД;Code:(.*);File:(.*);Severity:(.*);Log DateTime: (.*)	54 p_support 218f0b8a-834d-0e74-b42c-aca4636cda97 LOG_CRC_MISMATCH_FILE $MountPoint/1/home/user/apmdz/log/ahsm_log LOG_SEVERITY_ALERT Mon Feb 17 10:06:12 2025
SNMP	1_3_6_1_4_1_9_9_41_1_2_3_1_2_3="LINK";1_3_6_1_4_1_9_9_41_1_2_3_1_3_3=4;1_3_6_1_4_1_9_9_41_1_2_3_1_4_3="UPDOWN";1_3_6_1_4_1_9_9_41_1_2_3_1_5_3="Interface FastEthernet0/2, changed state to up";1_3_6_1_4_1_9_9_41_1_2_3_1_6_3=(182210813)	1_3_6_1_4_1_9_9_41_1_2_3_1_2_3="LINK";1_3_6_1_4_1_9_9_41_1_2_3_1_3_3=(.*);1_3_6_1_4_1_9_9_41_1_2_3_1_4_3="UPDOWN";1_3_6_1_4_1_9_9_41_1_2_3_1_5_3="(.*)";1_3_6_1_4_1_9_9_41_1_2_3_1_6_3=(.*)	4 Interface FastEthernet0/2, changed state to up (182210813)
WMI	Сведения 27.02.2025 12:58:56 Microsoft-Windows-Kernel-Boot 25 (32) «Использовалась следующая политика меню загрузки: 0x1.»	Сведения (.*)Microsoft-Windows-Kernel-Boot 25 (.*) \"(.*)\"	27.02.2025 12:58:56 (32) Использовалась следующая политика меню загрузки: 0x1.

одного часа) и обработки ЖС алгоритмом генерации РВ. Для моделирования внештатной ситуации на 77-м часе эксперимента была произведена искусственная DdoS-атака [26] на один из компьютеров ВС. В результате произошел наблюдаемый всплеск регистрируемых ЖС — 739 681. Как видно из результатов эксперимента (рис. 6–8), алгоритм построения РВ справился с обработкой многократно возросшего количества ЖС в реальном времени. Также в рамках эксперимента в ВС 3 периодически добавлялись отдельные устройства и компьютеры, однако они не оказали видимого влияния на результаты.

На графике (см. рис. 7) показан постоянный рост количества зарегистрированных в ВС ЖС, при этом уникальные шаблоны практически перестают накапливаться после пятого часа. Влияния DdoS-атаки с 77-го по 79-й час на формирование уникальных шаблонов не наблюдается. Это объясняется тем, что хотя ВС продуцирует большое количество ЖС во время атаки, они однотипны, поэтому уникальных шаблонов создается немного на фоне общего их числа.

Данные на графике (см. рис. 8) для показательности прологарифмированы. Точки, где нет новых шаблонов, обнаруженных в течение часа, выколоты. Небольшие всплески между накоплением шаблонов в начале и DdoS-атакой, а также после нее объясняются регистрацией редких ЖС и добавлением в ВС 3 отдельных устройств в ходе эксперимента.

Дальнейшие исследования показали, что наращивание ресурсов ВС не накладывает ограничений на обработку ЖС в реальном масштабе времени за счет распределенных вычислений.

Примеры структуризации ЖС различных форматов при помощи РВ, построенных в результате апробации разработанного алгоритма для ВС 1, ВС 2 и ВС 3, представлены в табл. 2.

В результате апробации алгоритма в контурах ВС 1, ВС 2 и ВС 3 все генерируемые ЖС были обработаны без потерь, вне зависимости от конфигурации источников, формата и частоты появления. Долгосрочных временных задержек при обработке не наблюдалось.

Заключение

В отличие от существующих научных решений [2–4, 6–24], а также коммерческих решений (см. табл. 1), предложенный алгоритм построения регулярных выражений для журнальных сообщений произвольных вычислительных систем позволяет структурировать ЖС (выделять характеристики) в реальном масштабе времени независимо от их (ЖС) формата, набора характеристик и частоты встречаемости в ВС.

Проведенные исследования алгоритма на реальных ВС показали, что вертикальное масштабирование, горизонтальное масштабирование или их комбинация позволяют обрабатывать ЖС в реальном времени для ВС любой конфигурации.

Структурированные данные ЖС могут быть использованы в анализаторах аномалий и компьютерных атак в ВС, актуальны для расследований инцидентов информационной безопасности и для аудита ВС.

Литература

1. **Zhu J., He S., Liu J., He P., Xie Q., Zheng Z., Lyu M. R.** Tools and benchmarks for automated log parsing. *Proceedings — 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice, ICSE-SEI*, 2019. doi:10.48550/arXiv.1811.03509
2. **Драчев Г. А.** Разработка алгоритма выделения и кодирования данных из журнальных сообщений вычислительной системы для систем обнаружения аномалий. *Информационные технологии*, 2023, т. 29, № 7, с. 351–359. doi:10.17587/it.29.351-359
3. **Савицкий Д. Е., Дунаев М. Е., Зайцев К. С.** Выявление аномалий при обработке потоковых данных в реальном времени. *International Journal of Open Information Technologies*, 2022, т. 10, № 6, с. 70–76.
4. **Du M., Li F., Zheng G., Srikumar V.** Deeplog: Anomaly detection and diagnosis from system logs through deep learning. *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 1285–1298. doi:10.1145/3133956.3134015
5. **Shendi M. M., Elkadi H. M., Khafagy M. H.** A study on the big data log analysis: goals, challenges, issues, and tools. *International Journal of Soft Computing and Artificial Intelligence*, 2019, vol. 7, pp. 1–12.
6. **Абрамов А. Г.** Высокопроизводительный сервис сбора и анализа файлов журналов сетевого и серверного оборудования в национальной исследовательской компьютерной сети. *Программные продукты и системы*, 2024, т. 37, № 4, с. 495–503. doi:10.15827/0236-235X.148.495-503
7. **Хасанова А. М.** Интеллектуальный анализ процессов по данным журналов событий информационных систем. *International Journal of Open Information Technologies*, 2022, т. 10, № 10, с. 70–77.
8. **Костиков Е. В.** Методы анализа логов Sysmon для обнаружения киберугроз. *International Journal of Open Information Technologies*, 2024, т. 12, № 11, с. 25–34.
9. **Гумеров Б. З.** Методы обогащения событий информационной безопасности с помощью CRIBL и

MISP. *Проблемы современной науки и образования*, 2022, № 6 (175), с. 38–45.
10. **Li Z., Fu Q., Huang Z., Yu Y., Lai Y., Ma Y.** Revisiting log parsing: The present, the future, and the uncertainties. *IEEE Transactions on Reliability*, 2024, pp. 1–14. doi:10.1109/TR.2023.3340020
11. **Chen S., Liao H.** BERT-log: Anomaly detection for system logs based on pre-trained language model. *Applied Artificial Intelligence*, 2022, vol. 36, no. 1, pp. e21456422028014. doi:10.1080/08839514.2022.2028014
12. **Hu J., Long L., Sui H., Gu Z., Zheng G.** CFTL: System log parsing method driven from clustering according to first token and length for anomaly detection. *Applied Sciences*, 2025, vol. 15, pp. 1740. doi:10.3390/app15041740
13. **Jin Bo Chen, Wen Yu Hu, Kang Hui Ying, Guo Nong Li.** A log analysis technology based on FP-growth improved algorithm. *2021 International Conference on Artificial Intelligence. Big Data and Algorithms (CAIBDA*), Xi'an, 2021, 28–30 May, 2021, pp. 219–223. doi:10.1109/CAIBDA53561.2021.00053
14. **Xiao T., Quan Z., Wang Z., Zhao K., Liao X., Huang H., Du Y., Li K.** LPV: A log parsing framework based on vectorization. *IEEE Transactions on Network and Service Management*, 2023, pp. 1. doi:10.1109/TNSM.2023.3248124
15. **Chen X., Wang P., Chen J., Wang W.** AS-parser: Log parsing based on adaptive segmentation. *Proceedings of the ACM on Management of Data*, 2023, vol. 1, pp. 1–26. doi:10.1145/3626719
16. **Wei M., Wen J., He S., Xie K., Liang W., Xie G., Li K., Zhu Z.** TCMS: A multi-sequence log parsing method based on token conversion. *IEEE Transactions on Dependable and Secure Computing*, 2024, pp. 1–18. doi:10.1109/TDSC.2024.3520628
17. **Yu S., He P., Chen N., Wu Y.** Brain: Log parsing with bidirectional parallel tree. *IEEE Transactions on Services Computing*, 2023, pp. 1–12. doi:10.1109/TSC.2023.3270566
18. **Чаругин В. В., Чаругин В. В., Чесалин А. Н., Ушкова Н. Н.** Конструктор блоков обработки естественного языка и применение его в задаче структурирования логов в информационной безопасности. *International Journal of Open Information Technologies*, 2024, т. 12, № 9, с. 111–119.
19. **Liu Y., Wu Y., Song W., Chen Z., Li Z.** LogPrompt: Prompt engineering towards zero-shot and interpretable log analysis. *Proceedings of the 2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings (ICSE)*, 2024, pp. 364–365. doi:10.1145/3639478.3643108
20. **Zhang C., Xu W., Liu J., Zhang L., Liu G., Guan J., Zhou Q., Zhou S.** LogBase: A large-scale benchmark for semantic log parsing. *Proceedings of the ACM on Software Engineering*, 2025, vol. 2, pp. 2091–2112. doi:10.1145/3728969
21. **Jiang Z., Liu J., Chen Z., Li Y., Huang J., Huo Y., He P., Gu J., Lyu M.** LILAC: Log parsing using LLMs with adaptive parsing cache. *Proceedings of the ACM on Software Engineering*, 2024, vol. 1, pp. 137–160. doi:10.1145/3643733
22. **Liu S., Yun L., Nie S., Zhang G., Li W.** IPLog: An efficient log parsing method based on few-shot learning. *Electronics*, 2024, vol. 13, pp. 3324. doi:10.3390/electronics13163324
23. **Cheng H., Ying S., Duan X., Yuan W.** DLLog: An online log parsing approach for large-scale system. *International Journal of Intelligent Systems*, 2024, pp. 1–17. doi:10.1155/2024/5961993
24. **Rücker N., Maier A.** FlexParser—The adaptive log file parser for continuous results in a changing world. *Journal of Software: Evolution and Process*, 2022, pp. 34. doi:10.1002/smr.2426
25. **Маркин Д. И., Гортинский А. В.** Использование технологии WMI для сбора информации и отслеживания событий в ОС Windows. *Информационная безопасность регионов*, 2016, № 4 (25), с. 11–15.
26. **Jun J.-H., Oh H., Kim S.-H.** DDoS flooding attack detection through a step-by-step investigation. *2011 IEEE 2nd International Conference on Networked Embedded Systems for Enterprise Applications*, Perth, 2011, 8–9 December, 2011, pp. 1–5. doi:10.1109/NESEA.2011.6144944

UDC 004.492.3
doi:10.31799/1684-8853-2026-1-36-47
EDN: LSYLOP

Algorithm for automatic construction of regular expressions for preprocessing arbitrary-format log messages in computing systems

G. A. Drachev[a,b], Researcher, Visiting Lecturer, Project Manager, orcid.org/0000-0003-1851-5507, pendal2@gmail.com
[a]National Research University Higher School of Economics, 34, Tallinskaya St. 123458, Moscow, Russian Federation
[b]Special System Engineering Center — Service, 71, Varshavskoe Highway, 117556, Moscow, Russian Federation

Introduction: Preprocessing is necessary to structure log messages by extracting characteristic fields for subsequent analysis aimed at detecting anomalies and cyber attacks in the computing system. The most acceptable approach, which dominates commercial solutions, is the construction of regular expressions corresponding to log messages. However, creating these regular expressions requires labor-

intensive manual processing. **Purpose:** To develop an algorithm for the automatic construction of regular expressions for log messages in real time, applicable to any log message regardless of its format or source. **Results:** The analysis of log message sources, their delivery methods to storage and processing systems, and existing log message formats has shown that log messages, even within a single format, often lack standardization in field composition. We have developed an algorithm to generate a template corresponding to a given log message and, through template processing, construct a regular expression with extracted fields effectively structuring the log message. Additionally, we have designed a system for storing accumulated log message templates and their corresponding regular expressions. For experimental research, we have developed a software toolkit to automatically construct regular expressions from log message texts. The toolkit has been tested on real computing systems with various configurations. **Practical relevance:** The proposed algorithm enables structuring of log messages of arbitrary types and formats. Structured log messages can be used for cybersecurity incident investigations, information system audits, and as input data for anomaly and cyberattack analyzers.

Keywords — information security, system logs, log messages, data preprocessing, log message structuring, regular expression.

For citation: Drachev G. A. Algorithm for automatic construction of regular expressions for preprocessing arbitrary-format log messages in computing systems. *Informatsionno-upravliaiushchie sistemy* [Information and Control Systems], 2026, no. 1, pp. 36–47 (In Russian). doi:10.31799/1684-8853-2026-1-36-47, EDN: LSYLOP

References

1. Zhu J., He S., Liu J., He P., Xie Q., Zheng Z., Lyu M. R. Tools and benchmarks for automated log parsing. *Proceedings — 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice, ICSE-SEI*, 2019. doi:10.48550/arXiv.1811.03509
2. Drachev G. A. Development of an algorithm for extracting and encoding data from log messages of a computing system for anomaly detection systems. *Information Technologies*, 2023, vol. 29, no. 7, pp. 351–359 (In Russian). doi:10.17587/it.29.351-359
3. Savitsky D. E., Dunaev M. E., Zaytsev K. S. Anomaly detection in real-time streaming data processing. *International Journal of Open Information Technologies*, 2022, vol. 10, no. 6, pp. 70–76 (In Russian).
4. Du M., Li F., Zheng G., Srikumar V. Deeplog: Anomaly detection and diagnosis from system logs through deep learning. *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 1285–1298. doi:10.1145/3133956.3134015
5. Shendi M. M., Elkadi H. M., Khafagy M. H. A study on the big data log analysis: goals, challenges, issues, and tools. *International Journal of Soft Computing and Artificial Intelligence*, 2019, vol. 7, pp. 1–12.
6. Abramov A. G. High-performance service for collecting and analyzing network and server hardware log files on a National Research Computer Network. *Software & Systems*, 2024, vol. 37, no. 4, pp. 495–503 (In Russian). doi:10.15827/0236-235X.148.495-503
7. Khasanova A. M. Process mining methods to analyze event logs of information systems. *International Journal of Open Information Technologies*, 2022, vol. 10, no. 10, pp. 70–77 (In Russian).
8. Kostikov E. V. Sysmon log analysis methods for cyber threat detection. *International Journal of Open Information Technologies*, 2024, vol. 12, no. 11, pp. 25–34 (In Russian).
9. Gumerov B. Z. Methods for the enrichment of infomation security events using CRIBL and MISP. *Problemy sovremennoj nauki i obrazovaniya*, 2022, no. 6 (175), pp. 38–45 (In Russian).
10. Li Z., Fu Q., Huang Z., Yu Y., Lai Y., Ma Y. Revisiting log parsing: The present, the future, and the uncertainties. *IEEE Transactions on Reliability*, 2024, pp. 1–14. doi:10.1109/TR.2023.3340020
11. Chen S., Liao H. BERT-log: Anomaly detection for system logs based on pre-trained language model. *Applied Artificial Intelligence*, 2022, vol. 36, no 1, pp. e21456422028014. doi:10.1080/08839514.2022.2028014
12. Hu J., Long L., Sui H., Gu Z., Zheng G. CFTL: System log parsing method driven from clustering according to first token and length for anomaly detection. *Applied Sciences*, 2025, vol. 15, pp. 1740. doi:10.3390/app15041740
13. Jin Bo Chen, Wen Yu Hu, Kang Hui Ying, Guo Nong Li. A log analysis technology based on FP-growth improved algorithm. *2021 International Conference on Artificial Intelligence. Big Data and Algorithms (CAIBDA)*, Xi'an, 2021, pp. 219–223. doi:10.1109/CAIBDA53561.2021.00053
14. Xiao T., Quan Z., Wang Z., Zhao K., Liao X., Huang H., Du Y., Li K. LPV: A log parsing framework based on vectorization. *IEEE Transactions on Network and Service Management*, 2023, pp. 1. doi:10.1109/TNSM.2023.3248124
15. Chen X., Wang P., Chen J., Wang W. AS-parser: Log parsing based on adaptive segmentation. *Proceedings of the ACM on Management of Data*, 2023, vol. 1, pp. 1–26. doi:10.1145/3626719
16. Wei M., Wen J., He S., Xie K., Liang W., Xie G., Li K., Zhu Z. TCMS: A multi-sequence log parsing method based on token conversion. *IEEE Transactions on Dependable and Secure Computing*, 2024, pp. 1–18. doi:10.1109/TDSC.2024.3520628
17. Yu S., He P., Chen N., Wu Y. Brain: Log parsing with bidirectional parallel tree. *IEEE Transactions on Services Computing*, 2023, pp. 1–12. doi:10.1109/TSC.2023.3270566
18. Charugin V. V., Charugin V. V., Chesalin A. N., Ushkova N. N. Constructor of natural language processing blocks and its application in the problem of structuring logs in information security. *International Journal of Open Information Technologies*, 2024, vol. 12, no. 9, pp. 111–119 (In Russian).
19. Liu Y., Wu Y., Song W., Chen Z., Li Z. LogPrompt: Prompt engineering towards zero-shot and interpretable log analysis. *Proceedings of the 2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings (ICSE)*, 2024, pp. 364–365. doi:10.1145/3639478.3643108
20. Zhang C., Xu W., Liu J., Zhang L., Liu G., Guan J., Zhou Q., Zhou S. LogBase: A large-scale benchmark for semantic log parsing. *Proceedings of the ACM on Software Engineering*, 2025, vol. 2, pp. 2091–2112. doi:10.1145/3728969
21. Jiang Z., Liu J., Chen Z., Li Y., Huang J., Huo Y., He P., Gu J., Lyu M. LILAC: Log parsing using LLMs with adaptive parsing cache. *Proceedings of the ACM on Software Engineering*, 2024, vol. 1, pp. 137–160. doi:10.1145/3643733
22. Liu S., Yun L., Nie S., Zhang G., Li W. IPLog: An efficient log parsing method based on few-shot learning. *Electronics*, 2024, vol. 13, pp. 3324. doi:10.3390/electronics13163324
23. Cheng H., Ying S., Duan X., Yuan W. DLLog: An online log parsing approach for large-scale system. *International Journal of Intelligent Systems*, 2024, pp. 1–17. doi:10.1155/2024/5961993
24. Rücker N., Maier A. FlexParser—The adaptive log file parser for continuous results in a changing world. *Journal of Software: Evolution and Process*, 2022, pp. 34. doi:10.1002/smr.2426
25. Markin D. I., Gortinsky A. V. Using WMI technology to collect data and trace events in Windows operating system. *Informacionnaja bezopasnost' regionov*, 2016, no. 4(25), pp. 11–15 (In Russian).
26. Jun J.-H., Oh H., Kim S.-H. DDoS flooding attack detection through a step-by-step investigation. *2011 IEEE 2nd International Conference on Networked Embedded Systems for Enterprise Applications*, Perth, 2011, pp. 1–5. doi:10.1109/NESEA.2011.6144944