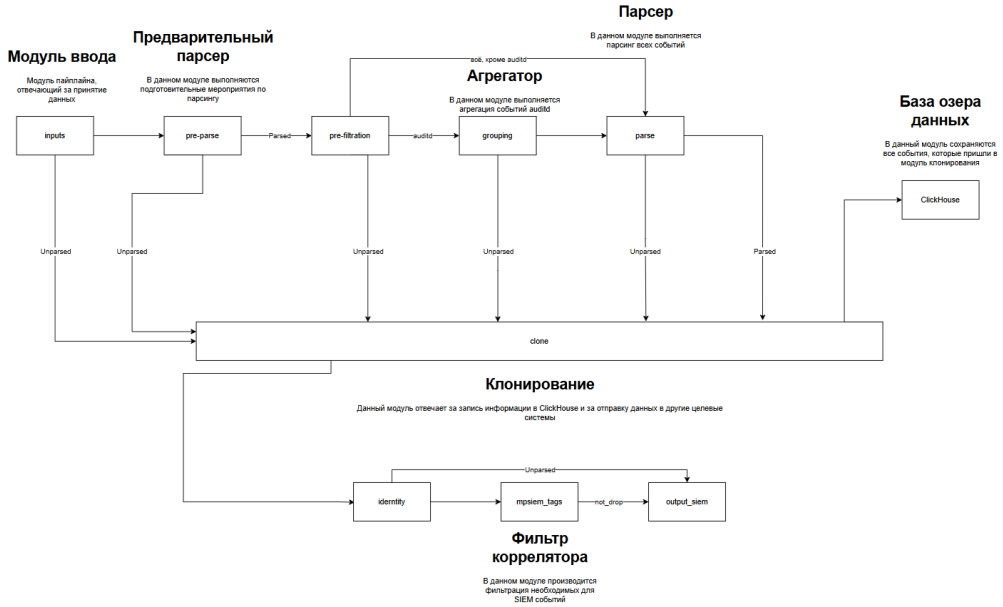
Обработка событий в Saturn

В данной статье описана схема обработки событий и блоки, которые выполняют обработку событий.



На рисунке выше показана схема обработки событий в компоненте **Saturn**. Каждый из блоков представляет собой определенную сущность в Системе.

Устройство конфигурационных файлов

Каждый конфигурационный файл состоит из трех секций:

- **input {}** — секция, описывающая источник данных;
- **filter {}** — секция, описывающая инструкции по изменению и преобразованию событий;
- **output {}** — секция, описывающая направление для данных, прошедших все этапы в блоке.

Блок **filter {}** обычно используется для небольших условий. В случаях, когда требуется выполнить фильтрацию по большому количеству параметров, или используется сложная логика фильтрации, рекомендуется вынести правила фильтрации в отдельный файл внутри директории.

Блок input

Данный блок является единой точкой входа всех событий в обработку. В нем описываются источники, из которых приходят данные, а также задаются последующие маршруты для принятых событий. В стандартной конфигурации события с тегом **unparsed** отправляются в блок **clone**, а с тегом **parsed** — в блок **pre_parse**.

Пример секции input

```
input {
  syslog_parallel {
     port => 5140
     type => "syslog"
     id => "syslog_1"
     grok_pattern => "(?<message>\<(?<priority>\d+)\>.*)"
     workers => 32
  }
}
```

Пример секции filter

```
filter {
  if [type] == "syslog" {
    mutate {
      add_field => {
        "recv_ipv4" => "%{host}"
        "recv_time" => "%{@timestamp}"
      }
    }
    if " audit" in [message] and ": [ID " not in [message] and [message] =~ /\s+auditd?(\[\d+\])?:\s+(?!\[ID\s+)/
        or "data-prepper" in [message] and [message] =~ /\s+data-prepper(\[\d+\])?:/
    {
      mutate {
        add_tag => ["unparsed", "custumer_skip"]
      }
    }
    else
    {
      # define product by message
      ruby {
        init => '
          # Предварительно кэшированные слова
          @product_map = {
            "auditd" =>  {regex: /\s+(audispd|audisp-.+?)(\[?\d+\])?:/.freeze, keyword: "audisp"},
             ...
            "unix_like" => {regex: /(\s|>)(?:login(:|\[\d+\])|sshd|klish|init|unix_chkpwd|sudo|dpkg|yum|RpmDb|rlogind|lightdm|su:|passwd|pkexec|kernel:|rshd|in.rexecd|systemd|root:|shutdown|usermod|useradd|userdel|vr
          }

        '
        code => '
          message = event.get("message")
          matched = false

          # Проверка каждого элемента с предопределенными ключевыми словами
          @product_map.lazy.each do |product, regex_data|
            if message.include?(regex_data[:keyword]) && message =~ regex_data[:regex]
              event.set("[product]", product)
              matched = true
              return
            end
          end

          event.tag("unparsed") unless matched
        '
      }
    }
  }
```

```
}
```

Пример секции output

```
output {
  if "unparsed" in [tags] {
    pipeline {
      send_to => clone
   }
  }
  else {
    pipeline {
      send_to => pre_parse
   }
  }
}
```

Блок pre_parse

Данный блок осуществляет предварительный парсинг (подготовка к дальнейшей обработке) поступивших событий. Например, в нем формируются дополнительные поля (msg) и выполняется более тонкая фильтрация событий на обработку (проверяются условия на соответствие парсерам). В стандартной конфигурации события с тегом **unparsed** отправляются в блок **clone**, а с тегом **parsed** — в блок **pre_filtration**.

Пример секции input

```
input {
  pipeline {
    address => pre_parse
  }
}
```

Пример секции filter

```
filter {

    if [product] == "auditd" {

  }

    # FILE: ../pipeline/pre_parse/checkpoint.filter.conf
    else if [product] == "checkpoint_cef" {
    grok {
      match => {
        "message" => "(:?<%{NUMBER}>)?CEF:0\|Check Point\|%{DATA:event_src_subsys}\|Check Point\|%{GREEDYDATA:msg}"
       }
    }
    if ![event_src_subsys] {
      mutate {
        add_tag => "unparsed"
      }
    }
  }
}
```

Пример секции output

```
output {
  if "unparsed" in [tags] {
    pipeline {
      send_to => clonning
    }
  }
  else {
    pipeline {
      send_to => pre_filtration
    }
  }
}
```

Блок pre_filtration

Данный блок осуществляет распределение событий в блоки в соответствии со значением поля **product** (в блоке **parse**). В стандартной конфигурации события с тегом **product = auditd** отправляются в блок **grouping**, события с тегом **unparsed** отправляются в блок **clone**. Таким образом, можно выделить три направления для событий: нормализатор соответствующего продукта, агрегатор **grouping** для логов auditd и clone для тех событий, которые не смогли классифицировать.

Пример секции input

```
input {
  pipeline {
    address => pre_filtration
  }
}
```

Пример секции filter

Для данного блока не предусмотрена дополнительная фильтрация.

Пример секции output

```
output {
  if "unparsed" in [tags] {
    pipeline {
      send_to => clonning
    }
  }
  else {
    if [product] == "auditd" {
      pipeline {
        send_to => grouping
      }
    }
    else if [product] == "ms_dns" {
      pipeline {
        send_to => ms_dns
      }
    }
  }
}
```

Блок grouping

Данный блок выполняет группировку (агрегацию) событий **auditd**. В стандартной конфигурации события с тегом **parsed** отправляются в блок **parse**, события с тегом **unparsed** отправляются в блок **clone**.

Пример секции input

```
input {
  pipeline {
    address => grouping
  }
}
```

Пример секции filter

```
filter {
  if [product] == "auditd" {
    # check for auditd aggregate/grouping option
    grok {
      match => [ "message", "(?<source>audispd|audisp).+?(node=%{NOTSPACE:node}\s+)?type=%{NOTSPACE:aggtype}\s+msg=audit\(%{INT:aggtimestamp}[.,]%{INT:timestampfractional}\:%{INT:eventid}\)\:\s*%
      tag_on_failure => [ "_not_for_aggregation", "_grokparsefailure"]
    }

    # aggregate/group auditd events
    if "_not_for_aggregation" not in [tags] {
      aggregate {
        task_id => "%{node}%{eventid}%{timestampfractional}%{timestamp}"
        code => "
             map['host'] ||= event.get('host')
             ...
             event.cancel()
         "
        push_map_as_event_on_timeout => true
        timeout => 30 # 10 minutes timeout
        # проставляем теги агрегированности и отправки в SIEM
        timeout_tags => ['_aggregated']
      }
    }
  }
  if "_aggregated" in [tags] {
      # set mime type for aggregated field
      mutate {
        add_field => {
          "mime" => "application/json"
          "product" => "auditd"
        }
      }
      # build body string from json object
      ruby {
        code=>"
          require 'json'
          b = event.get('body')
          event.set('message', JSON.generate(b))
        "
      }
    }
}
```

Пример секции output

```
output {
  if "unparsed" in [tags] {
    pipeline {
      send_to => clonning
    }
  }
  else {
      pipeline {
        send_to => parse
    }
  }
}
```

Блок parse

Данный блок является совокупностью блоков, каждый из которых обрабатывает события только для своего **product**. После окончания процедуры парсинга, все события отправляются в пайплайн **clone**.

Пример секции input

```
input {
  pipeline {
    address => parse
  }
}
```

Пример секции filter

Для данного блока не предусмотрена дополнительная фильтрация.

Пример секции output

```
output {
  pipeline {
    send_to => clonning
  }
}
```

Блок clone

Данный блок осуществляет отправку всех событий в хранилище (ClickHouse), выполняет предобработку событий Windows, а также отправляет события в пайплайн identity, если включена фильтрация на основе правил корреляции.

После прохождения данного блока изменения событий в базу LogIQ записаны не будут.

Пример секции input

```
input {
  pipeline {
    address => clonning
  }
}
```

Пример секции filter

Для данного блока не предусмотрена дополнительная фильтрация.

Пример секции output

```
output {
  if "${MPSIEM_ENABLED:false}" {
     pipeline {
       send_to => identity
     }
    }

  clickhouse {
    id => "output.clickhouse"
    urls => "${CLICKHOUSE_URLS}"
    username => "${CLICKHOUSE_USER}"
    password => "${CLICKHOUSE_PASSWORD}"
    database => "${CLICKHOUSE_DB}"
    table => "${CLICKHOUSE_TABLE}"
    mutations => {
      "uuid" => "uuid"
      ...
      "tags" => "tags"
    }
  }
}
```

Блок identity

Данный блок отвечает за распределение событий по тегам. Например, в стандартной конфигурации, если в событиях **product = checkpoint_gaia/auditd/usergate/postgresql/cisco/esxi/nginx/unix_like/vsphere**, события отправляются в блок **mpsiem_tags**. В ином случае события отправляется сразу в блок **output_siem**.

Пример секции input

```
input {
  pipeline {
    address => identity
  }
}
```

Пример секции filter

```
filter {
   if [type] == "syslog" {
      if "unparsed" in [tags] and [host] ==  "122.29.20.120"  {
        drop {}
      }
      if ([facility] == 0 or [severity] == 7) and ([host] != "15.228.181.11" and [host] != "10.228.147.85" and [host] != "172.30.1.172")    {
        drop {}
      }
      if ([host] == "122.29.20.125" or [host] == "172.19.123.250" or [host] == "15.228.18ц.11" or [host] == "10.218.19.125" or [host] == "15.228.181.11") and [message] =~ /\/sdk/ {
        drop {}
      }
   }
}
```

Пример секции output

```
output {
    if "unparsed" in [tags] {
      pipeline {
        send_to => output_siem
      }
    }
   else {
      pipeline {
        send_to => mpsiem_tags
      }
    }
}
```

Блок mpsiem_tags

Данный блок является системой фильтрации событий согласно правилам корреляции взятым из SIEM. Для корректной работы важно найти нужное правило корреляции и удостовериться, что событие подпадает под фильтр, а в конце ставится тег **not_drop**. Именно по нему, как правило, события уходят на отправку. Все остальные события отбрасываются.

Пример секции input

```
input {
  pipeline {
    address => mpsiem_tags
  }
}
```

Пример секции filter

Для данного блока не предусмотрена дополнительная фильтрация, так как правила берутся напрямую из SIEM.

Пример секции output

```
output {
   if "not_drop" in [tags] {
    pipeline {
      send_to => output_siem
    }
   }
    else if "kaspersky_security_for_linux_mail_server" == [event_src_title] {
     pipeline {
      send_to => output_siem
    }
   }
}
```

Блок output_siem

Финальный блок, отвечающий за отправку данных в SIEM. В случае с MP SIEM данные отправляются сразу брокеру, минуя все механизмы ingestion.

Пример секции input

```
input {
  pipeline {
    address => output_siem
  }
}
```

Пример секции filter

Для данного блока не предусмотрена дополнительная фильтрация, так как он является финальным.

Пример секции output

```
output {
      rabbitmq {
        id => "output.siem.1"
        host => "${SIEM_RMQ_HOST}"
        user => "${SIEM_RMQ_USER}"
        password => "${SIEM_RMQ_PASSWORD}"
        exchange => "${SIEM_RMQ_EXCHANGE}"
        exchange_type => "${SIEM_RMQ_EXCHANGE_TYPE}"
        key => "${SIEM_RMQ_KEY}"
        vhost => "${SIEM_RMQ_VHOST}"
        batch_publish => "${SIEM_RMQ_BATCH_ON}"
        batch_size => "${SIEM_RMQ_BATCH_SIZE}"
      }
}
```

ID статьи: 445
Последнее обновление: 30 апр., 2025
Обновлено от: Егоров В.
Ревизия: 5
База знаний LogIQ -> Документация -> Система хранения и обработки данных «LogIQ». Версия 2.4.0 -> LogIQ. Руководство разработчика -> Управление конфигурационными файлами -> Обработка событий в Saturn
https://docs.axel.pro/entry/445/