Платформа Радар

Описание специальных функций корреляции

Версия 4.2.0

ООО «Пангео Радар»

Оглавление

1. Описание редактора Lua для правил корреляции

1.1. Структура правила

По умолчанию, при создании правила корреляции, его текст имеет вид:

```
local detection_windows = "10m"
local create_incident = false
local assign_to_customer = false
local risk_score = 2
local grouped_by = {}
local aggregated_by = {}
local grouped_time_field = "@timestamp"
local template = ""

function on_logline(logline)
  log("accept logline")
  -- meta = {}
  -- incident_identifier = logline:get("event.field", "")
  -- asset = get_fields_value(logline:raw(), {"target.host.ip",
"target.host.fqdn", "target.host.hostname"})
  -- alert({
      -- template = template,
      -- risk_level = risk_score,
      -- asset_ip = asset[1],
      -- asset_hostname = asset[2],
      -- asset_fqdn = asset[3],
      -- asset_mac = "",
      -- create_incident = create_incident,
      -- assign_to_customer = assign_to_customer,
      -- logs = {loglines},
      -- meta = meta,
      -- incident_identifier = incident_identifier
  -- })
end

-- function on_grouped(grouped)
--
-- end
```

Вверху находится блок с переменными, которые отвечают за настройку группера и срабатывания правила (`create_incident, assign_to_customer`).

Функция `on_logline` всегда должна быть в правиле, вызывается каждый раз коррелятором при поступлении логлайна, соответствующего фильтрам правила.

Параметр `logline` позволяет обращаться к текущему логлайну и имеет два метода:

- `logline:raw()` – возвращает строку в котором содержится логлайн (json);
- `logline:get(path, default)` – получить значения поля логлайна, например `logline:get("initiator.fqdn", “”);`
- `logline:get_fields(path_array, [{defaults}])` – получить значения полей (см. get_fields_value)

- `logline:decode()` – преобразует логлайн в объект, что позволяет обращаться к полям напрямую.

  Пример:

  ```
  ll = logline:decode() -- декодируем логлайн в объект (таблица)
  log(ll.event.field) -- выводим содержимое поля event.field в лог
  ```

  Примечание: операция более медленная, чем обращение через `logline:get(…)`.

- `logline:get_asset_data(path)` – получает значение поля логлайна по пути, в отличие от get в случае если значение по заданному пути является массивом вернет его первый элемент или пустую строку, если массив пустой.

1.2. Грунперы

Существует два вида грунперов: стандартный и pattern matcher. Все функции стандартного так же доступны для pattern matcher, но не наоборот.

Определение стандартного группера:

```
grouper1 = grouper.new(
    grouped_by,
    aggregated_by,
    grouped_time_field,
    detection_windows,
    on_grouped
)
```

Где:

- `grouped_by` – группировка по полям;
- `aggregated_by` – по каким полям агрегировать;
- `grouped_time_field` – описание поля, содержащего время в логлайне, а также формата времени (следует после запятой).

Пример:

```
"event.dt,2006-01-02 15:04:05"
"@timestamp,UnixMilli"
```

Если передать пустую строку, то в качестве времени логлайна будет использовано текущее время.

Возможные описания формата времени:

Строка формата	**Пример строки с датой**
RFC3339Nano	2006-01-02T15:04:05.999999999Z07:00
RFC3339	2006-01-02T15:04:05Z07:00

Строка формата	Пример строки с датой
ANSIC	Mon Jan _2 15:04:05 2006
UnixDate	Mon Jan _2 15:04:05 MST 2006
RubyDate	Mon Jan 02 15:04:05 -0700 2006
RFC822Z	02 Jan 06 15:04 -0700
RFC850	Monday, 02-Jan-06 15:04:05 MST
RFC1123	Mon, 02 Jan 2006 15:04:05 MST
RFC1123Z	Mon, 02 Jan 2006 15:04:05 -0700
Kitchen	3:04PM
Stamp	Jan _2 15:04:05
StampMilli	Jan _2 15:04:05.000
StampMicro	Jan _2 15:04:05.000000
StampNano	Jan _2 15:04:05.000000000
UnixMilli	Число, содержащее UNIX время в миллисекундах
UnixMicro	Число, содержащее UNIX время в микросекундах

`detection_windows` – окно жизни событий (логлайнов) в группере. формат: число со строчным суффиксом.

Возможные суффиксы:

Суффикс	Величина времени
ms	Миллисекунды
s	Секунды
m	Минуты
h	Часы

`on_grouped` – функция, вызываемая при срабатывании группера. Данная функция в скрипте правила должна объявляться ранее, чем создание группера.

Сам объект группера (grouper1) содержит два метода:

Метод	Описание
grouper1:countAgg(массив_строк)	Вызов дополнительной группировки для конкретного поля. Параметр принимает массив имен полей по которым требуется сгруппировать. Возвращает словарь (Dict) для каждого поля: {fields_key = {field_value = count}}
grouper1:clear()	“Очищает” группер. Помечает логлайны участвующие в текущей(!) группировке как “использованные”, чтобы они не попадали больше в группировку и не вызывали дублирование

Примечание: очистка не требуется для группера типа pattern matcher, очистка в этом случае выполняется автоматически.

Пример функции on_grouped:

```
function on_grouped(grouped)

    log("agg total: "..grouped.aggregatedData.aggregated.total.." for hash key "..grouped.key)

    if grouped.aggregatedData.aggregated.total >= 5 then -- and grouped:first("path") == "value"

        -- check custom grouper
        resTmp = grouper1:countAgg({"target.ip"})
        check_ok = false
        for k, data in pairs(resTmp) do
            for keyCount, count in pairs(data) do
                -- log("key: " .. k .. ", key count" .. keyCount .. ", count" .. count)

                if k == "172.30.254.30__4000" and keyCount == "172.30.254.30" and count == 2 then
                    check_ok = true
                end
            end
        end
        if not check_ok then
            error("count check failed")
        end
```

Продолжение примера на следующей странице:

```
        asset = get_fields_value(grouped.aggregatedData.loglines[1],
{"target.ip", "target.hostname", "target.fqdn"})

        meta = {var = 123}

        alert({
            template = template,
            risk_level = 0.5,
            asset_ip = asset[1],
            asset_hostname = asset[2],
            asset_fqdn = asset[3],
            asset_mac = "",
            create_incident = true,
            assign_to_customer = false,
            logs = grouped.aggregatedData.loglines,
            meta = meta,
            incident_identifier = ""
        })

        grouper1:clear()
    end
end
```

В функцию `on_grouped` передается параметр `grouped`, в котором содержатся данные группера.

Описание данных:

Поле	**Тип**	**Описание**
grouped.key	Строка	“ключ” группера, собранные в одну строку значения полей группировки
grouped.groupedFields	Массив строк	Массив полей группировки
grouped.aggregatedData.loglines	Массив строк	Массив логлайнов, которые участвовали в группировке
grouped.aggregatedData.aggregated.count	Объект	Поля и их счетчики, пример: {"agg_field_1": count, ...}. Где agg_field_1 имя поля агрегации
grouped.aggregatedData.aggregated.total	Число	Сумма счетчиков
grouped.aggregatedData.aggregated.countByField	Объект	Поля агрегации и их значения со счетчиками, пример: {"agg_field_1": [{"agg_value": count},...], ...}. Где agg_field_1 имя поля агрегации, agg_value - значение поля агрегации
grouped.aggregatedData.unique.data	Объект	Уникальные значения полей агрегации, пример: {"agg_field_1": ["unique_value"], ...}. Где agg_field_1 имя поля агрегации, unique_value - уникальное значение поля агрегации
grouped.aggregatedData.unique.count	Объект	Уникальные счетчики по полям агрегации, пример: {"agg_field_1": count, ...}. Где agg_field_1 имя поля агрегации
grouped.aggregatedData.unique.valuesByField	Объект	Уникальные значения по полям агрегации

Определение pattern matcher:

```
pattern = {
    { field = "action", values = {"detect"}, count = 1 },
    { field = "action", values = {"delete", "clean", "quarantine"}, absent = true
},
}

grouper1 = grouper.new_pattern_matcher(
    {"target.file.path", "target.host.ip", "target.threat.name"},
    {},
    {"@timestamp"},
    pattern,
    "@timestamp",
    detection_windows,
    on_matched
)
```

Формат записи паттерна:

```
{ field = "имя поля", values = массив_значений, count = счетчик_повторов [,
(опционально) absent = true] }
```

где `absent` - флаг, указывающий на то, что значения не должно быть.

Использование флага `absent` делится на три возможных варианта:

- absent в начале - означает, что срабатывание произойдет, если в указанном окне (detection_windows) будет найдено совпадение по pattern'у И не будет значений absent в начале.
- absent в середине - обычное сравнение, где проверяется отсутствие указанных значений во всем паттерне.
- absent в конце - означает, что срабатывание произойдет, если в указанном окне (detection_windows) будет найдено совпадение по pattern'у И не будет значений absent в конце.

Функция коллбэк отличается от стандартного группера:

```
function on_matched(grouped, matchedData)
    log("on_matched, key: " .. grouped.key .. " matched data len: " ..
table.getn(matchedData.loglines))
    return true

end
```

`grouped` - стандартный объект группера (описание выше), к нему добавляется поле `matchedData`, массив всех срабатываний группера.

`matchedData` - объект, описывающий текущий pattern match. Описание полей:

Поле	**Тип**	**Описание**
matchedData.loglines	Массив строк	Логлайны соответствующие настройкам pattern'а

Функция `on_matched` должна возвращать `true`, если требуется вернуть следующие срабатывания (pattern match'и). Если все матчи обрабатываются за раз (с помощью `grouped**.**matchedData`), то функция должна вернуть `false`.

1.3. Массивы

Пример	**Описание**
grouped.aggregatedData.loglines[1]	Получить первый элемент (логлайн)
grouped.aggregatedData.loglines[#grouped.aggregatedData.loglines]	Получить последний элемент (логлайн)
map (функция, массив)	Возвращает массив с произведенной операцией описанной в функции. Пример: function inverse(item) return not item end res = map(inverse, {true, false}) вернет res = {false, true}
any (массивбулевскихзначений)	Вернет true, если хоть один из элементов массива = true
contains (массив, значение)	Возвращает true, если хоть один элемент массива равен значению

1.4. Функции

1.4.1 Работа со строками

Пример	**Описание**
string.len(“строка“) или (”строка”):len()	Возвращает длину строки
string.join("разделитель", массив_строк) или ("разделитель"):join(массив_строк) или table.concat(массив, "разделитель")	Объединение массива строк в строку с разделителем
string.sub(“строка”, начало) или string.sub("abc", начало, конец)	Возвращает подстроку, если не указан конец, то от начала до конца строки
("строка"):trim()	Убирает whitespaces (пробел, перевод строки) из строки
("строка”):split("разделитель")	Разбивает строку с разделителем на массив строк
("строка"):search("^(http	ftp)s?:")
("строка"):endswith(“подстрока“)	Возвращает true, если строка оканчивается на подстроку
("строка"):startswith("подстрока")	Возвращает true, если строка начинается на подстроку
("строка"):upper()	Возвращает строку, переведенную в верхний регистр
("строка"):lower()	Возвращает строку, переведенную в нижний регистр

1.4.2 Работа с логлайнами (json в строке)

Пример	Описание
get_field_value(логлайн, “путь”)	Получить значение в пути. Путь - ссылка на поле json, например target.ip, будет соответствовать {“target”: {“ip” : “значение“}}. Более подробно можно посмотреть тут
get_fields_value(логлайн, массив_путей)	Возвращает массив значений из логлайна. Пример: asset = get_fields_value(grouped.aggregatedData.loglines[1], {"target.ip", "target.hostname", "target.fqdn"})
set_field_value(логлайн, "путь", значение)	Устанавливает значение в логлайне по указанному пути и возвращает измененный логлайн. Пример: my_logline = set_field_value(my_logline, "new_field", 123). Примечание: логлайном может быть так же массив логлайнов, тогда для каждого объекта в массиве будет установлено значение, в этом случае замена происходит прямо в переданном массиве (не требуется получать возвращаемое значение)

1.4.3 Отладка

Пример	Описание
sleep(миллисекунды)	“Засыпает” на указанное количество миллисекунд
log(значение)	Выводит сообщения о работе правила в журнал `rule.log`, который располагается по пути `/var/logs/rule-logs/<id-правила>`. Значением может быть строка, число, объект или булевый тип. Параметры сообщений в журнале правила: - формат события: `{"level":"info","message":"world","function":"log","time":"2025-02-20T16:37:28+03:00"}`; - уровень журналирования: debug, info, warn, error; - `function` - источник (функция правила) сообщения. например "log"; - `time` - время; - `event_id` - опциональное поле с идентификатором события.
set_debug_value(имя, значение)	Устанавливает значение отладочной переменной, выводится в результатах тестирования
error(строка)	Вызвать ошибку в правиле с описанием “строка”

Для того, чтобы уменьшить избыточность логирования на потоке, одинаковые сообщения группируются, выводятся только первые два сообщения (затем возможен повтор через какое то время, в зависимости от кол-ва повторяемых сообщений и количества уникальных сообщений). Если обязательно требуется записывать каждое (например, из правила с помощью функции log), то следует добавить какой-либо счетчик в текст сообщения. Например:

```
counter = 0
function on_logline(logline)
  log("accept logline " .. tostring(counter))
  counter = counter + 1
end
```

1.4.4 Табличные списки (RVS)

Обращение к табличным спискам происходит с помощью вызова глобальной функции `storage.new("имя_справочника")`, пример:

```
test_storage = storage.new("test")
```

Далее работа идет с переменной хранилища:

Имя метода	**Описание**
test_storage:id()	Возвращает идентификатор табличного списка
test_storage:set(“ключ”, “имя_колонки“, “значение“ [, (опционально) TTL])	Устанавливает значение по ключу для указанной колонки, если задано TTL (в миллисекундах) то устанавливается время жизни ключа (текущее время + указанное TTL)
test_storage:get(“ключ“, “имя_колонки”)	Возвращает значение по ключу для указанной колонки
test_storage:set_values("ключ", {value = "string value", num = 123} [, (опционально) TTL])	Устанавливает сразу несколько значений по ключу для выбранных колонок. Где value - имя колонки (используется по умолчанию) типа строка, и num - имя колонки с типом число
test_storage:get_values(“ключ”)	Возвращает все значения (всех колонок) по ключу. Например: values = test_storage:get_values(“test”) log(values.value .. “ “ .. values.num)
test_storage:remove(“ключ“)или если требуется удалить несколько test_storage:remove({“ключ1“, “ключ2“})	Удаляет ключ (или перечень ключей) и его значения
test_storage:count()	Получить количество записей в справочнике
test_storage:truncate()	Стирает все данные из справочника
test_storage:search("имя_колонки", "val1")	Ищет заданное значение в колонке с указанном именем во всем справочнике, возвращает имя первого ключа, если значение нашлось, иначе nil
test_storage:search_all("имя_колонки", "значение")	Ищет заданное значение в колонке с указанном именем во всем справочнике, возвращает список всех ключей с указанным значением. Значение может быть формата “LIKE”, а именно: %str - ищем значения, заканчивающиеся на str; %str% - ищем значения с подстрокой str; str% - ищем значения, начинающиеся со str
test_storage:calc(список_ключей, "имя_колонки")	Выполняет калькуляцию по указанным ключам по указанной колонке. Возвращает объект с полями count, errors, min, max, avg, sum. Errors - количество ошибок (приведение типов). Имя_колонки - если пустое, то используется имя по умолчанию (“value”)
test_storage:check_ip(“имя_колонки_cidr“, “ip_address“)	Проверяет вхождение IP (ip*address) в подсети указанные в табличном списке (имя*колонки_cidr)
test_storage:key({ip = "127.0.0.1", host = “comp_name“})	Возвращает подсчитанный из значений колонок “ключей” идентификатор записи. Где ip и host это имена колонок “ключей”. В параметрах должны быть указаны все колонки “ключи” табличного списка. Пример использования: -- получить значение колонки count для записи с ip = 127.0.0.1 и host = localhost -- ip и host в табличном списке являются ключами test_storage:get(test_k_storage:key({ip="127.0.0.1", host="localhost"}), "count") -- удалить строку с ip = 127.0.0.1 и host = localhost test_storage:remove(test_k_storage:key({ip="127.0.0.1", host="localhost"}))

1.4.5 Память правила

Пример	Описание
memory.set("имя_переменной", “значение", TTL)	Устанавливает значение имени переменной в памяти с указанным TTL. TTL - время жизни в миллисекундах, считается от текущего времени, если указано 0, хранится все время жизни правила (до отключения или перезагрузки правила
memory.get("имя_переменной")	Возвращает значение имени переменной, или nil, если значение не найдено (или время жизни переменной истекло)

1.4.6 Математика

Пример	Описание
sum(массив_чисел) или sum(объект, “поле”)	Сумма всех элементов массив. Пример по сумме в объекте: sum({{test = 1}, {test = 2}}, "test")
avg(массив_чисел)	Среднее значение всех элементов массива

1.4.7 Вспомогательные функции

Пример	Описание
is_home_net(строка_с_ip_адресом)	Проверяет входит ли ip адрес в домашнюю сеть, подсети задаются в конфигурации сервиса logmule
is_home_net_arr(массив_строк_с_ip)	Проверяет входят ли все ip адреса в домашнюю сеть
in_any_network(строка_с_ip_адресом, массив_с_подсетями)	Проверяет входит ли ip любую из указанных сетей. Пример: in_any_network("192.168.1.1", {"172.0.0.0/8", "192.168.0.0/16"})
event_register(объект, массив_с_идентификатором_правил)	Отправляет объект(логлайн) в очередь указанных правил на корреляцию
now_in_ms()	Возвращает локальное текущее время в миллисекундах (UnixMilli)
type(значение)	Возвращает тип значения: ”bool” - булевое значение, ”number” - число, ”string” - строка, ”nil” - пустое, ”function” - функция, ”table” - массив, ”user” - внутренний объект
uuid()	Возвращает сгенерированный UUID (строка)

2. Описание специальных функций для правил корреляции

Правило корреляции представляет из себя небольшой скрипт на языке Python.

Благодаря гибкости языка **Python** Платформа поддерживает широкий список базовых функций. В том числе:

- операции равенства значений, строкового равенства (независимо от регистра значений);
- операции неравенства значений, строкового неравенства (независимо от регистра значений);
- операции больше, больше или равно, меньше, меньше или равно;
- поиск неполного значения;
- сравнение поля (значения) за временной диапазон, фиксация изменений;
- проверка, что значение в поле начинается с определенного значения;
- проверка наличия или отсутствия значения в поле (поле пустое/не пустое);
- проверка, что значение из поля входит или не входит в указанный список или списки;
- проверка наличия или отсутствия определённого поля в событии;
- использование переменных, для хранения промежуточных значений корреляции;
- сравнение полей (значений) между различными полями события;
- конкатенация (склеивание) значений различных полей внутри одного события;
- использование отрицания к определенному условию или группе условий.

2.1. Синтаксис правил корреляции

2.1.1 Основные правила

Базовый функционал Python который необходимо соблюдать:

- Отступы
- Табуляция или пробелы
- Пустые строки
- Использование пробелов
- Соглашения по именованию
- Декораторы
- Метод get()

Пример простого правила, создающего инцидент для каждого события, удовлетворяющего критериям отбора:

```
# Регистрация подписки на события
@log_connection.fetch('#.microsoft.windows.os.authentication.#')
def handle_logline(logline):
    # проверка значений полей событий
    if logline.action == "login_fail" and logline.user == "Admin":
        # Создание инцидента
        alert("admin_login", logline, 8.0, logline.asset_info,
              create_incident=True, assign_to_customer=True)
```

2.1.2 Поддержка UTF-8

Кодировка ASCII и UTF-8 поддерживается в данных объектов `logline`, но не в именах правил и ключей маршрутизации.

Строки UTF-8 должны быть определены как строка **utf** (для этого добавляется «u» слева, например, `u"fiёld"`), в случае отсутствия пометки код не выведет предупреждение или ошибку и строки не будут совпадать.

Пример:

```
@log_connection.fetch("ascii-text-only-here")
def handle_logline(line):
    _val = line.get(u"fiёld", None) # field не найдет ни одного
элемента
    if _val == u"한글":
        print("Matched")
        alert(..)
    elif _val == "한글": # u отсутствует
        print("Не найдет совпадений!")
    else:
        print("Не совпадает или отсутствует")
```

2.2. Основной функционал правил корреляции

2.2.1 Ключ маршрутизации `routing key`

Ключ маршрутизации представляет собой последовательность из следующих значений, разделенных точкой:

- event.category;
- event.subcategory;
- action;
- outcome (если задано; в противном случае заменяется на “none”);
- reason. (если задано; в противном случае заменяется на “none”);
- “none” (для обратной совместимости с предыдущими версиями продукта);
- “none” (для обратной совместимости с предыдущими версиями продукта);
- “none” (для обратной совместимости с предыдущими версиями продукта);
- logsource.vendor;
- logsource.product;
- logsource.application;
- logsource.subsystem.

Примеры:

```
application.dns_answer.answer.success.dns/answer/noerror.none.none.none.suricata.
suricata.suricata.suricata.dns


authentication.group_member_added.modify.success.none.none.none.none.microsoft.wi
ndows.os.authentication
```

2.2.2 Объект `logline`

События, которые правило получает из очереди, приходят в виде объекта `logline`, атрибутами которого являются поля нормализованного события.

Пример:


```
{
  "routing_key":"#.kaspersky-sc.mssql.security_center.#",
  "@timestamp":"2020-02-12T14:35:01.242532+00:00",
  "epoch":1581518101.242532,

  "event":{
    "severity":5.0,
    "category":"antivirus",
    "subcategory":"test.pgr.local",
    "description":"description",
    "worker":{"ip":"172.18.0.27", "host":"cd72b9351328"},
    "logsource":{
      "subsystem":"security_center",
      "product":"kaspersky-sc",
      "vendor":"kaspersky",
      "name":"KES",
      "application":"mssql",
      "host":"172.18.0.27",
      "input":"kaspersky-sc"
    },
    "uuid":"6632bcc38b24cde9bae0ed18d0ec43c8"
  }

  "initiator":{
    "host":{"ip":["10.30.25.123"], "hostname":["test2"],
"fqdn":["test.pgr.local"], "geoip":[]    },
    "user":{"domain":"test.pgr.local","name":"test2\\\\testuser2"}
  },
  "target":{
    "service":{"name":"security/av"},
    "threat":{"name":"malware"},
    "file":{

"path":["C:\\\\Users\\\\usefulgarbage\\\\Desktop\\\\av72\\\\eicar_com.zip//eicar.c
om"],
      "hash":{"sha256":["\u003csha256 HASH object @ 0x7f6dd8462be8\u003e"]},
      "name":["Usefulgarbage test signature"]
    }
  },

  "action":"detect_malware",
  "outcome":{
    "description":"Результат:     Обнаружено: EICAR-Test-File\\r\\nПользователь:
pgrdc01\\\\btsymzhitov (Активный пользователь)\\r\\nОбъект:
C:\\\\Users\\\\user\\\\Desktop\\\\av72\\\\eicar_com.zip//eicar.com\\r\\nПричина:
Экспертный анализ\\r\\nДата выпуска баз:     6/20/2019 1:51:00 AM\\r\\nХеш:
275a021bbfb6489e54d471899f7db9d1663fc695ec2fe2a2c4538aabf651fd0f\\r\\n"
  },

  "observer":{
    "host":{"ip":["10.30.25.1"],"hostname":[],"fqdn":[]    },
    "event":{"type":"GNRL_EV_VIRUS_FOUND","id":"9979744"    }
  },
```


Продолжение примера на следующей странице:

```
  "reportchain":{
    "collector":{
      "timestamp":"2020-02-12T17:35:01.000000+03:00",
      "host":{"ip":["10.10.100.2"],"hostname":[],"fqdn":[]}
    },
    "relay":{
      "timestamp":"2020-02-12T17:35:01.000000+03:00",
      "host":{"ip":["10.30.25.1"],"hostname":[],"fqdn":[]}
    }
  },

  "raw":"<...>",
}
```

К полям объекта `logline` можно обращаться:

- как к атрибуту объекта `logline.имя_поля`;
- как к полю словаря `logline["имя_поля"]`.

Допускается обращение ко вложенным объектам. Пример:

```
data = {
    "@timestamp": "2015-01-01 01:00:00.123456+01:00",
    "@epoch": 1420070400.123456,
    "src": "1.2.3.4",
    "geo_info": {
        "country": "Far Far off",
        "position": {
            "lat": 10.0,
            "long": 20.0,
        }
    }
}
logline = Logline(data)
logline.ts
# datetime.datetime(2015, 1, 1, 1, 0, 0, 123456, tzinfo=tzoffset(None, 3600))
logline.epoch
# 1420070400.123456
logline["src"]
# "1.2.3.4"
logline.geo_info.country
# "Far Far off"
logline.geo_info.color_of_magic
# None
logline.bogus
# None
logline.bogus.even_worse
# Ошибка AttributeError
logline["geo_info.country.position.lat"]
# 10.0
```

2.2.3 Отладочная печать значений `print`

Для целей отладки доступна функция `print`, позволяющая выводить значения переменных в стандартный поток вывода.

Примечание: *большое количество вызовов `print` негативно влияет на производительность правил – рекомендуется использовать только для отладки*.

2.2.4 Регистрация подписки на события `fetch`

Для объявления подписки на события из очереди событий на корреляцию используется метод `fetch`, после чего объявляется функция `handle_logline()`, которая принимает на вход полученное событие.

Пример:

```
@log_connection.fetch('#.microsoft.windows.os.authentication.#')
def handle_logline(logline):
    # вывод в отладочную консоль
    print(logline)
    # дальнейшая обработка события

    # ...
```

В качестве параметра передается шаблон для `routing key`. В шаблоне допускается использовать `wildcard #`.

Одна функция может регистрировать несколько подписок на события.

Пример:

```
@log_connection.fetch('#.microsoft.windows.os.system.#')
@log_connection.fetch('#.microsoft.windows.os.authentication.#')
@log_connection.fetch('#.suricata.suricata.#')
def handle_logline(logline):
    # вывод в отладочную консоль
    print(logline)
    # дальнейшая обработка события

    # ...
```

2.2.5 Регистрация результата или инцидента `alert`

При вызове функции `alert` формируется результат работы правила, который записывается в базу данных и отображается в интерфейсе Платформы.

Функция позволяет автоматически создавать инциденты по результатам работы правила и назначать его на ответственную группу пользователей.

Формат вызова функции `alert`:

```
alert(
    template_name,
    logline,
    risklevel,
    asset_info,
    create_incident=False,
    assign_to_customer=False,
    incident_identifier=None

)
```

Где:

- **template_name** (str) — имя связанного шаблона.
- **logline** (Logline или AggregatedLogline) — результаты работы функций корреляции и связанные события.
- **risklevel** (float или int) — уровень риска, присваиваемый данному результату (0.0 – 10.0).
- **asset_info** (dict) — словарь значений для поиска актива в базе, может состоять из следующих ключей:
    - **ip (обязательный) –** IP-адрес актива;
    - **hostname –** имя хоста;
    - **mac –** MAC-адрес хоста;
- **create_incident** (bool, по умолчанию False) — автоматическое создание инцидента по данному результату. Инцидент будет создан в статусе «Отладка».
- **assign_to_customer** (bool, по умолчанию False) —автоматическое назначение инцидента группе ответственных. Инцидент будет создан в статусе «Новый».
- **incident_identifier** (str) – уникальный идентификатор в рамках типа инцидента для объединения с нужным происшествием (Например имя учетной записи).

Пример:

```
@grouper.register()
def handle_grouped(logline):
    asset = build_asset(logline.observer.host.ip,  # IP-адрес актива
                        logline.observer.host.fqdn,  # MAC-адрес хоста
                        logline.observer.host.hostname)  # имя хоста

    alert("template",  # имя связанного шаблона
          logline,  # (Logline или AggregatedLogline - grouped, matched) – результаты работы функций корреляции и связанные события
          rule_settings["risk_score"],  # уровень риска, присваиваемый данному результату (0.0 – 10.0)
          asset,  # словарь значений для поиска актива в базе
          create_incident=rule_settings["create_incident"],  # автоматическое создание инцидента по данному результату
          assign_to_customer=rule_settings["assign_to_customer"],   # автоматическое назначение инцидента группе ответственных
          incident_identifier=logline.initiator.user.name)         #    уникальный идентификатор в рамках типа инцидента для объединения с нужным происшествием
```

В результат, помимо заданных полей, можно вставлять дополнительные поля. Значение дополнительного поля будет отображаться в карточке инцидента, связанного с результатом

обработки правила корреляции. Подробнее о работе с дополнительными полям см. руководство «**Пользователь**» → «**Описание интерфейса**» → «**Инциденты».**

Для добавления дополнительного поля добавьте строку в правило корреляции `alert`:

```
custom_values={
  "user"="domain\username"
  }
```

Пример:

```
alert("template", logline, 9.0, asset_info,
create_incident=True,
      custom_values={
            "user"="domain\username"
      }
)
```

В данном примере "user" - ключ дополнительного поля, "domain\username" - значение дополнительного поля.

2.2.6 Настройки правила `rule_settings`

Настройки правила — это глобальная переменная (доступная в любом месте правила), которая объявляется в начале правила с указанием параметров необходимых для работы правила:

```
rule_settings = {
    # основные
    "detection_windows": "5m",  # один или несколько размеров окна группировки
    "risk_score": 8.0,  # оценка риска при создании алерта
    "create_incident": False,  # автоматическое создание инцидента по данному
результату
    "assign_to_customer": False,  # автоматическое назначение инцидента группе
ответственных, инцидент будет создан в статусе «Новый»
    # дополнительные
    "RAW_lines_to_store": 2,  # количество сообщений отображаемых в алерте,
используется совместно с функцией trim_result
    "distinct_events": 2  # требуемое количество различных событий, используется
совместно с функцией EventChain
}
```

2.2.7 Связанные хранилища значений `stores`

Хранилища значений закрепляются за правилом в момент его создания и содержат в себе дополнительную информацию для работы правила:

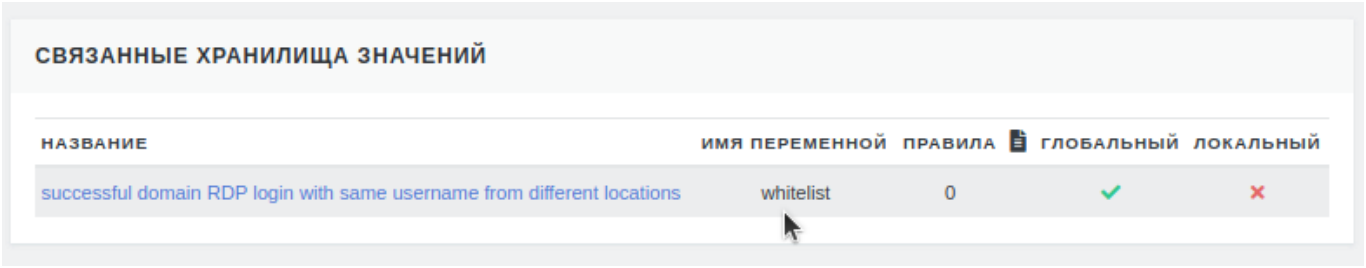

Рисунок 1 – Связанные хранилища значений

```
# username, uuid, location
[
  ['padmin', '9b1deb4d-3b7d-4bad-9bdd-2b0d7b3dcb6d', 'moscow']
  ['cadmin', 'c1c0a397-add5-475e-9ae6-70cb2e67561b', 'tyumen']
]
```

Доступ к хранилищу производится через объект `stores` по имени задекларированному при создании правила:

```
whitelist_username = [acc_data[0] for acc_data in stores["whitelist"]]  # сгенерируется список состоящий из username (['padmin', 'cadmin']), после чего будет присвоен whitelist_username
whitelist_uuid = [acc_data[1] for acc_data in stores["whitelist"]]  # сгенерируется список состоящий из uuid (['9b1deb4d-3b7d-4bad-9bdd-2b0d7b3dcb6d', 'c1c0a397-add5-475e-9ae6-70cb2e67561b']), после чего будет присвоен whitelist_uuid
whitelist_location = [acc_data[2] for acc_data in stores["whitelist"]]  # сгенерируется список состоящий из location (['moscow', 'tyumen']), после чего будет присвоен whitelist_location
```

2.2.8 Метод вставки `set`

Метод вставки позволяет добавлять необходимые поля из логлайна в результат работы функции корреляции. Вызов:

```
@grouper.register ()  # регистрация функции обработки результатов
def handle_grouped(grouped):  # зарегистрированная функция обработки результатов, которая в качестве параметра получает объект AggregatedLogline (часто используется параметр с именем grouped)
    all_loglines = grouped.loglines  # список объектов logline

    grouped.set("observer", all_loglines[-1].observer)  # из списка объектов logline берется последний элемент из которого достается значение observer и помещается в объект AggregatedLogline по ключу "observer"
    grouped.set("initiator", all_loglines[-1].initiator)  # из списка объектов logline берется последний элемент из которого достается значение initiator и помещается в объект AggregatedLogline по ключу "initiator"

    asset = build_asset(...)  # формирование ассета

    alert(...)  # функция регистрации результата или инцидента
```

2.2.9 Ограничение отображения количества сырых событий в алерте `trim_result`

Объявление и вызов (пример с использованием Grouper - функция агрегации событий):

```
@grouper.register()  # декоратор register()
def handle_grouped(grouped):  # функция обработки результатов
    risk_score = rule_settings["risk_score"]

    # Вызов функции ограничения количества сырых событий в алерте
    trim_result(
        grouped,  # получает объект AggregatedLogline (например grouped или matched)
        rule_settings["detection_windows"],  # окна группировки
        rule_settings["RAW_lines_to_store"]):  # количество сообщений отображаемых в алерте
```

2.2.10 Связанные шаблоны `template`

Связанные шаблоны закрепляются за правилом в момент его создания и содержат в себе дополнительную информацию для формирования результата анализа:

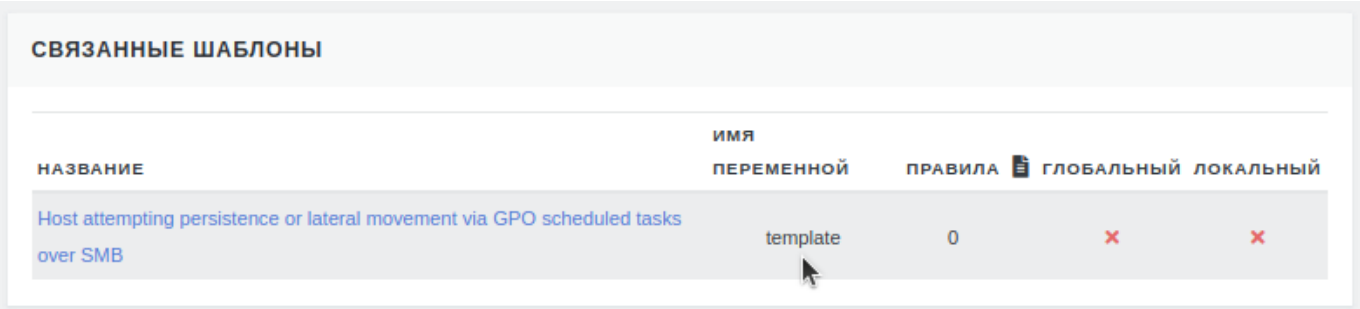

Рисунок 2 – Связанные шаблоны

```
Пользователь "{{ logline.initiator.user.name }}" (SID: {{
logline.initiator.user.id }}) с узла {{ logline.initiator.host.ip | join(", ") }}
удалённо изменил файл (ScheduledTasks.xml) групповой политики, отвечающего за
управление планировщиком задач Windows.
```

Доступ к связанному шаблону производится через имя шаблона, например `template`.

2.2.11 Запись результатов в файл event_register

Функция позволяет выполнить запись результатов работы правил корреляции в файл.

```
event_register({
            "finding_title": "Антивирус – Обнаружено вредоносное ПО",
            "asset_type": "Host",
            "logline_summary": matched.loglines,
            "result_asset_fqdn": matched_logline.target.host.fqdn,
            "result_description": "",
            "result_created_at": matched_logline.reportchain.collector.timestamp,
            "result_id": matched_logline.observer.event.id,
            "result_risk_impact": "",
            "result_incident_identifier": matched_logline.target.threat.name,
            "result_updated_at": matched_logline.reportchain.collector.timestamp,
            "rule_name": "AV-001-Malware detected and not removed Users",
            "result_occurred_at": matched_logline.reportchain.collector.timestamp,
            "correlated_asset_fqdn": matched_logline.target.host.fqdn,
            "result_title": "Обнаружено ВПО в пользовательском сегменте",
            "result_asset_ip": matched_logline.target.host.ip,
            "result_synopsis": "",
            "result_risklevel": rule_settings["risk_score"],
            "result_solution": "",
            "result_analysis_output": "",
            })
```

2.3. Агрегация событий

Применяется для поиска однотипных событий в рамках временного окна.

2.3.1 Базовая агрегация `Grouper`

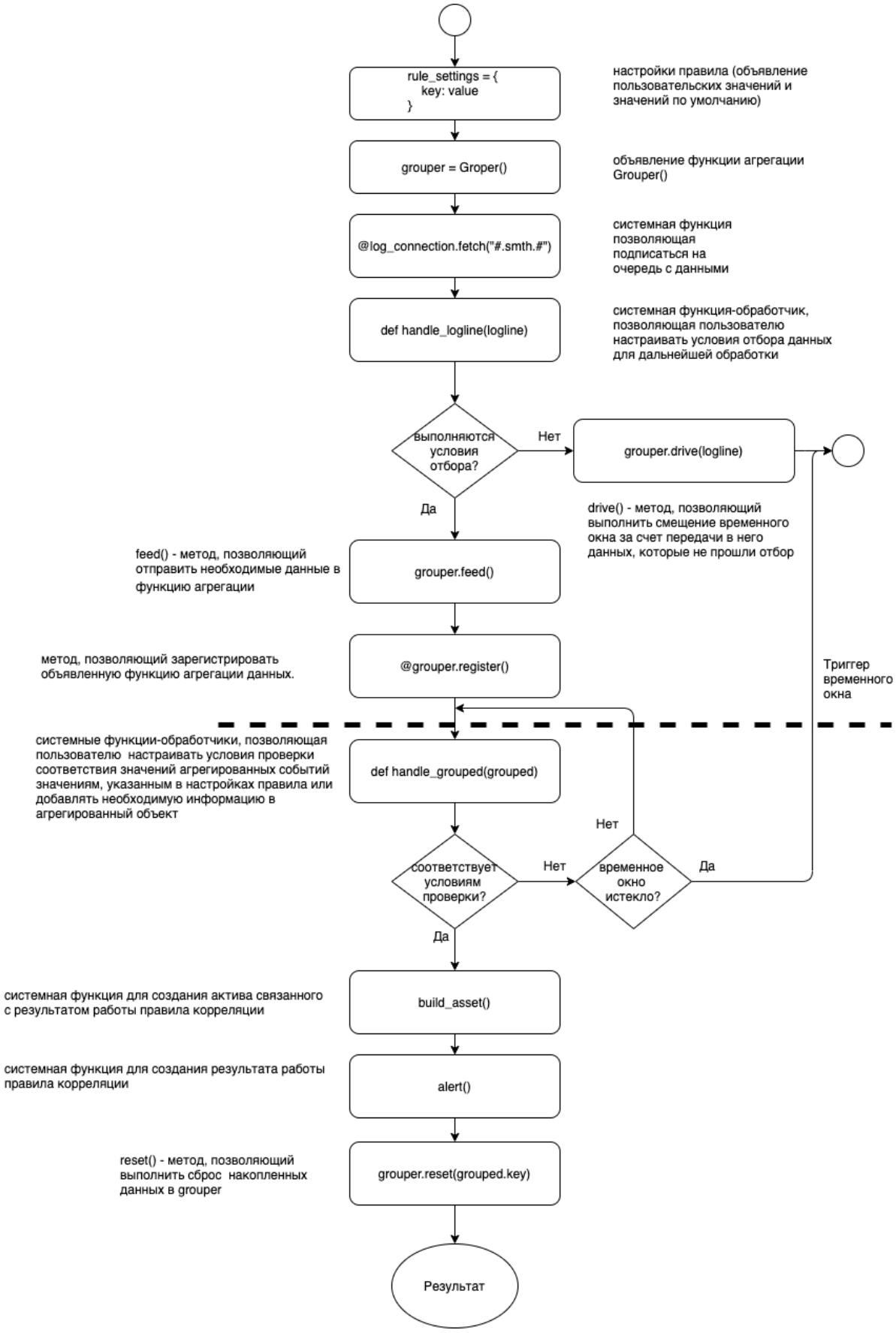

Рисунок 3 – Базовая агрегация grouper

Объявление функции агрегации событий:

```
grouper = Grouper(
      rule_settings["detection_windows"],  # размер окна группировки
      over=('initiator.user.name', 'initiator.user.id', 'observer.host.ip',
'observer.host.fqdn', 'observer.host.hostname'),  # поля в объектах logline, по
которым будет осуществляться группировка
      aggregation=Count(store_loglines=True)   # метод агрегации Count считающий
входящие события, параметр store_loglines который колекционирует входящие события
```

Регистрация обработчика

Функция-обработчик регистрируется с помощью метода `register`:

```
grouper = Grouper(["10m", "1h"], over=("src",))

@grouper.register()
def handle_grouped(grouped_logline):
    print(grouped_logline)
```

Значения из настроек правила

Временное окно - может быть массивом строк из двух значений или строкой из одного значения) Пример задания значений временного окна:

Массив значений, использующийся для скользящего окна внутри большого окна

```
rule_settings = {
      "detection_windows": ("1m","10m")
}
```

Одиночное значение

```
rule_settings = {
      "detection_windows": "10m"
}
```

Могут принимать значения:

- Секунду ("1s")
- Минуты ("1m")
- Часы ("1h")
- Дни ("1d")

over - определение уникальных значений из нормализованных событий, по которым производится подсчет группировка. Может быть одиночное имя поля или массив имен полей.

Пример:

По одному полю: `over=("initiator.host.ip")`

По двум и более: `over=("initiator.host.ip", "target.user.name")`

aggregation - способы агрегации данных.

Агрегация выполняется по полям, перечисленным в параметре over и позволяет выполнять следующие действия над данными:

- Count() - подсчет событий для сгруппированных полей, указанных в поле over. Может использоваться с дополнительные параметрами:
    - segment_by - перечень полей для формирования уникального ключа агрегации
    - store_loglines – необходимо сохранять события или нет. Доступные значения True и False.
- Collect() - получение уникального списка из указанного поля

Пример:
`grouper = Grouper("1h", over=("src.ip"), aggregation=Collect` (поле, по которому необходимо выполнить расчет))

- Sum() - суммирование данных указанного поля

Пример:
`grouper = Grouper("1h", over=("src.ip"), aggregation=Sum` (поле, по которому необходимо выполнить суммирование))

- Mean() - расчет среднего

Пример:

```
      return (logline.get("suricata.flow.bytes_sent", 0)
          - logline.get("suricata.flow.bytes_received"))
grouper = Grouper("1h", over=("src.ip"), aggregation=Mean(diff))
```

- Std() - расчет стандартного отклонения

Пример:

`grouper = Grouper("1h", over=("src.ip",), aggregation=Std` (поле, по которому необходимо выполнить расчет)

- Compound() – параллельный подсчет с применением разных функций подсчета

Пример:

```
aggregation=Compound(Sum("received_bytes"), Count()))
```

- clock – параметр отвечает за указание группперу, каким образои осуществлять смещение временного окна. Может принимать следующие значения
    - "logline" -смещение осуществляется с помощью метода drive() за счет объектов logline, которые не прошли отбор для агрегации. Значение по умолчанию.
    - "heartbeat" – системный счетчик, который отправляет системное сообщение в группер для принудительного (искусственного) смещения временного окна. Для работы функции необходимо в конце тела правила объявить системную функцию следующего вида:

```
def handle_heartbeat(heartbeat):
      grouper.feed(heartbeat)``
```

Результат работы агрегирующей функции

По окончанию временного окна вызывается зарегистрированная функция обработки результатов, которая в качестве параметра получает объект AggregatedLogline (часто используется параметр с именем `grouped`):

```
@grouper.register()
def handle_grouped(grouped):
    print(grouped)
```

Добавление событий в функцию агрегации производится с помощью метода `feed()`:

```
@log_connection.fetch('#.microsoft.windows.os.authentication.#')  # метод для
получения событий на корреляцию
def handle_logline(logline):  # функция принимающая полученные события на
корреляцию
     grouper.feed(logline)
```

Если корреляция строится по событиям, которые случаются редко, рекомендуется использовать сдвиг временного окна по данным из других событий. Для сдвига временного окна используется метод `drive()`:

```
@log_connection.fetch('#.microsoft.windows.os.authentication.#')  # метод для
получения событий на корреляцию
def handle_logline(logline):  # функция принимающая полученные события на
корреляцию
    if logline.event_id == "что-то редкое":
        grouper.feed(logline)
    else:
        grouper.drive(logline)
```

Если при объявлении группера указаны два окна, например, [“10m”, “1h”] функция-обработчик будет вызываться каждые 10 минут как для окна «10m», так и для окна «1h». Если указано `overlapping=False`, функция-обработчик будет вызываться каждые 10 минут для окна «10m» и каждый час для окна «1h».

```
grouper = Grouper(
      [“10m”, “1h”],  # размеры окон группировки
      over=('initiator.user.name', 'initiator.user.id', 'observer.host.ip',
'observer.host.fqdn', 'observer.host.hostname'),  # поля в объектах logline, по
которым будет осуществляться группировка
      aggregation=Count(store_loglines=True),  # метод агрегации
overlapping=False)
```

Состояние контекста агрегации может быть сброшено с помощью метода `reset()`:

```
grouper.reset()
```

Таким образом контекст будет переведен в исходное состояние до попадания в него объекта logline:

```
@grouper.register()
def handle_grouped(grouped):
    asset = build_asset(...)  # формирование ассета
    alert(...)  # функция регистрации результата или инцидента
    grouper.reset(grouped.key)
```

2.3.2 Сдвиг временного окна без добавления событий в функцию агрегации `drive`

Если корреляция строится по событиям, которые случаются редко, рекомендуется использовать сдвиг временного окна по данным из других событий.

Для сдвига временного окна используется функция `drive`:

```
grouper = Grouper(["5m"], handle_grouped)

@log_connection.fetch('#.microsoft.windows.os.authentication.#')
def handle_logline(logline):
    if logline.event_id == "что-то редкое":
        grouper.feed(logline)
    else:
        grouper.drive(logline)
```

2.3.3 Использование системного таймера для сдвига временного окна `heartbeat`

Для корреляции событий, которые приходят неупорядоченно или с некорректными полями @timestamp/epoch необходимо использовать системный таймер.

Использование системного таймера должно быть объявлено в функции группировки событий (`grouper`) — указать атрибут `heartbeat` в качестве атрибута часов (по умолчанию `logline`). Это приведет к тому, что агрегирование будет управляться внутренними часами, а не некорректными входящими полями времени.

Внутренний ключ маршрутизации **logmule.heartbeat** используется для получения специального элемента `logline` — `eartbeat` — который генерируется каждую секунду.

При передаче функции группировки событий (`grouper`) с помощью `heartbeat` вместо обычного `logline` функция группировки событий (`grouper`) использует временные поля `heartbeat` для управления процессом.

Пример:

```
# clock="heartbeat" – использование системного таймера для сдвига окна
grouper = Grouper("10s", over=("some-key",), clock="heartbeat")

# подписка на события, которые нас интересуют в рамках правила
@log_connection.fetch("#.microsoft.windows.os.authentication.#")
def handle_logline(logline):
    grouper.feed(logline)

# подписка на события системного таймера
@log_connection.fetch("logmule.heartbeat")
def handle_heartbeat(heartbeat):
    # свиг временного окна
    grouper.feed(heartbeat)
```

Для сохранения частоты для каждого правила должен быть только один обработчик `heartbeat`. Тем не менее `heartbeat` можно использовать более чем в одном вызове.

Пример:

```
@log_connection.fetch("some-route")
def handle_logline_a(logline):
    pass

@log_connection.fetch("different-route")
def handle_logline_b(logline):
    pass

@log_connection.fetch("logmule.heartbeat")
def handle_heartbeat(heartbeat):
    # Оба handle_logline_{a,b} получают heartbeat с периодом 1 секунда
    handle_logline_a(heartbeat)
    handle_logline_b(heartbeat)
```

2.3.4 Примеры использования функции группировки событий `grouper`

Пример:

```
grouper = Grouper(["10m", "1h"], over=("src",))
# Так как Count() является агрегацией по умолчанию, а единственное поле over
# не обязательно записывать как один кортеж, это эквивалентно следующему:
# grouper = Grouper(["10m", "1h"], over="src", aggregation=Count())

@grouper.register()
def handle_grouped(grouped):
    if grouped.value("10m") > 1000 or grouped.value("1h") > 6000:
        risklevel = 6.0
        if grouped.value("10m") > 1000 and grouped.value("1h") > 6000:
            risklevel += 2
        alert("dropped_connections", grouped, risklevel,
              logline.asset_info, create_incident=True)

@log_connection.fetch("firewall")
def handle_logline(logline):
    if (homenet(logline.src) and not homenet(logline.dst) and
        logline.action == "drop"):
        grouper.feed(logline)
```

Обратите внимание, что если поле, переданное в over, содержит списки, то необходимо учитывать порядок элементов.

Например, при группировке списка IP-адресов код `['1.1.1.1', '2.2.2.2']` создаст категорию, отличную от категории, заданной с помощью `['2.2.2.2', '1.1.1.1']`. Если такое поведение нежелательно, можно отсортировать список, прежде чем передавать его в функцию агрегации.

При указании нескольких размеров окна, например, [“10m”, “1h”] функция-обработчик будет вызываться каждые 10 минут как для окна «10m», так и для окна «1h». Если указано `overlapping=False`, функция-обработчик будет вызываться каждые 10 минут для окна «10m» и каждый час для окна «1h».

Состояние контекста агрегации может быть сброшено с помощью `grouper.reset()`. Таким образом контекст будет переведен в исходное состояние до попадания в него объекта `logline`.

Агрегации также могут работать со значениями попадающих в них `logline`. Значения могут быть объединены в результирующий список. Также можно выполнять различные операции над ними: суммировать, вычислять среднее, получать стандартное отклонение или процент. Для этого

необходимо передать в функцию группировки событий (`grouper) logline` либо объект, либо имя функции, возвращающей агрегируемое значение.

Создание списка, содержащего значения из всех объектов `logline` в рамках окна:

```
# Получения списка всех протоколов из обработанных flow
grouper = Grouper("1h", over=("src.ip",),
                  aggregation=Collect("suricata.flow.proto"))

# Суммирование значений поля somefield для всех logline в 1-часовом окне:
# (поле somefield должно быть числовым)
grouper1 = Grouper("1h", over="src.ip", aggregation=Sum("somefield"))

# Вычисление средней разницы между отправленными и полученными байтами:
def diff(logline):
    return (logline.get("suricata.flow.bytes_sent", 0)
            - logline.get("suricata.flow.bytes_received"))

grouper2 = Grouper("1h", over=("src.ip",), aggregation=Mean(diff))

# Вычисление стандартного отклонения по значениям поля somefield:
grouper3 = Grouper("1h", over=("src.ip",), aggregation=Std("somefield"))

# Получение списка процентилей (1 %, 5 %, 10 %, 50 %, 90 %, 95 %, 99 %) для
собранных значений:
grouper4 = Grouper("1h", over=("src.ip",),
                   aggregation=Percentile("somefield"))
```

2.3.5 Агрегация событий с использованием табличных списков

Все вспомогательные классы и объекты `logline` принимают не только доступ ко вложенным атрибутам, но и вызовы `__getitem__` с использованием точечной нотации.

Это позволяет использовать вложенные данные при агрегации, обучении и т.д.

Пример:

```
# Объвления функции агрегации
grouper = Grouper(["10m", "1h"], over=("asset_info.hostname",))

# Регистрация обработчика событий
@grouper.register()
def handle_grouped(grouped):
    pass # do something useful

@log_connection.fetch("firewall")
def handle_logline(logline):
    if (homenet(logline.src) and not homenet(logline.dst) and
        logline.action == "drop"):

        grouper.feed(logline)
```

2.3.6 Агрегирование по ключу

Функция группировки событий (`grouper`) может разбивать агрегаты по отдельному ключу (предоставляется `segment_by`).

Обратите внимание, что, если поле `segment_by` содержит списки, то порядок элементов имеет значение, поэтому, например, если необходимо разбить список IP-адресов, будет создан сегмент, отличный от созданного.

Если подобный вариант не подходит, можно отсортировать список перед передачей его в функцию группировки событий (`grouper`). `['1.1.1.1', '2.2.2.2']['2.2.2.2', '1.1.1.1']`

Пример:

```
# Объвления функции агрегации
grouper = Grouper(["10m", "1h"], over=("src.ip",),
                  aggregation=Count(segment_by=("dst.ip",)))
# Регистрация обработчика событий
@grouper.register()
def handle_grouped(grouped):
    grouped.value("10m")
    # == {"8.8.8.8": 20, "8.8.4.4": 7}
@log_connection.fetch()
def handle_logline(logline):

    grouped.feed(logline)
```

2.3.7 Агрегирование по нескольким ключам

Функция группировки событий (`grouper`) также может одновременно запускать несколько агрегаций в одном потоке `logline`.

Пример:

```
# Объвления функции агрегации
grouper = Grouper(["10m", "1h"], over=("src.ip",),
                  aggregation=Compound(Sum("received_bytes"), Count()))
# Регистрация обработчика событий
@grouper.register()
def handle_grouped(grouped):
    grouped.value("10m")
    # == (243568, 9)
@log_connection.fetch("bro_http")
def handle_logline(logline):

    grouper.feed(logline)
```

2.4. Работа с кросс-корреляциями

2.4.1 Контейнер для кросс-корреляций `CorrelationContainer`

Контейнер для кросс-корреляции между источниками. Позволяет обнаруживать цепочки последовательностей атак или последовательности событий, происходящих на одном и том же активе. Объявление функции корреляции:

```
CorrelationContainer(
    delta,
    distinct_limit,
    callback,
    clock

)
```

Где:

- **delta** (str) — окно корреляции.
- **distinct_limit** (int, по умолчанию "*2*") — требуемое количество различных категорий, которые должны быть отправлены в `callback`.
- **callback** (*callable*, по умолчанию *None*) — функция, вызываемая по завершению каждого временного окна. Если здесь передается None, то должен быть вызван метод `register` для регистрации функции обработки результатов.
- **clock** (str) — `logline` или `heartbeat`.

Пример:

```
correlation_container = CorrelationContainer("5m")
grouper = Grouper(["1m"], over=("src.ip",))
@correlation_container.register()
def handle_correlated(correlated)
    alert("kill_chain", correlated, 8.0, correlated.asset_info)
@log_connection.fetch("ids_events")
def handle_ids(logline):
    if logline.event.id in (12345, 56789):
        correlation_container.feed("binary download", logline,
                                   key=logline.generate_key("src.ip"))
@grouper.register()
def handle_grouped(grouped)
    if grouped.value("1m") > 200:
        correlation_container.feed("dns burst", grouped, window="1m")
```

2.4.2 Корреляция по цепочке событий `EventChain`

Функция корреляции различных событий детектирует заданное количество различных событий для заданного ключа в рамках временного окна.

Объявление:

```
event_chain = EventChain(
    rule_settings['detection_windows'],  # размер окна группировки
    rule_settings['distinct_events'])  # требуемое количество различных событий
```

Регистрация осуществляется с помощью декоратора `register()`:

```
@event_chain.register()
```

В случае успешной корреляции событий вызывается зарегистрированная функция обработки результатов, которая в качестве параметра получает объект `AggregatedLogline` (часто используется параметр с именем `matched`).

```
@event_chain.register()
def handle_matched_pattern(matched):
    print(matched)
```

Добавление событий в функцию корреляции производится с помощью метода `feed()`:

```
@log_connection.fetch('#.microsoft.windows.os.authentication.#')  # метод для
получения событий на корреляцию
def handle_logline(logline):  # функция принимающая полученные события на
корреляцию
     # Генерируется ключ, по которому определяется разница между событиями
     event_logline = logline.generate_key('initiator.host.ip',
'initiator.host.hostname', 'initiator.host.fqdn')
     # Определяется ключ, который должен быть идентичный в событиях
     event_key = logline.target.user.id
     event_chain.feed(event=event_logline, key=event_key, logline=logline)
```

Удаление событий из функции корреляции производится с помощью метода `cancel()`:

```
@log_connection.fetch('#.microsoft.windows.os.authentication.#')
def handle_logline(logline):
     # Генерируется ключ, по которому определяется разница между событиями
     event_logline = logline.generate_key('initiator.host.ip',
'initiator.host.hostname', 'initiator.host.fqdn')
     # Определяется ключ, который должен быть идентичный в событиях
     event_key = logline.target.user.id
     if logline.event.subcategory == 'vpn_open':
         event_chain.feed(event=event_logline, key=event_key, logline=logline)
     elif logline.event.subcategory == 'vpn_close':
         event_chain.cancel(
             event=event_logline,  #  удаляемое значение. Если задано None будут
удалены все значения для key
         key=event_key)  # ключ группировки
```

2.4.3 Корреляция событий по шаблону `PatternMatcher`

Возможно создавать проверку из нескольких `PatternMatcher`.

Объявление:

```
pattern_matcher = PatternMatcher(
    rule_settings['detection_windows'],  # размер окна группировки
    ['4624'] + ['4672'],  # требуемый шаблон для поиска
    order_by='@timestamp')  # поле по которому выполнять сортировку входящих
событий
```

Доступные параметры:

- order_by - поле из нормализованного сортировка по которому будет осуществляться проверка последовательности.
- Regex – включение директивы использования регулярных выражений в паттернах выявления последовательностей. Доступные значения `True` и `False`.

Регистрация осуществляется с помощью декоратора `register()`:

```
@pattern_matcher.register()
```

В случае успешной корреляции событий вызывается зарегистрированная функция обработки результатов, которая в качестве параметра получает объект `AggregatedLogline` (часто используется параметр с именем `matched`).

```
@pattern_matcher.register()
def handle_matched_pattern(matched):
    print(matched)
```

Схема корреляции событий по шаблону появления типов событий в рамках временного окна:

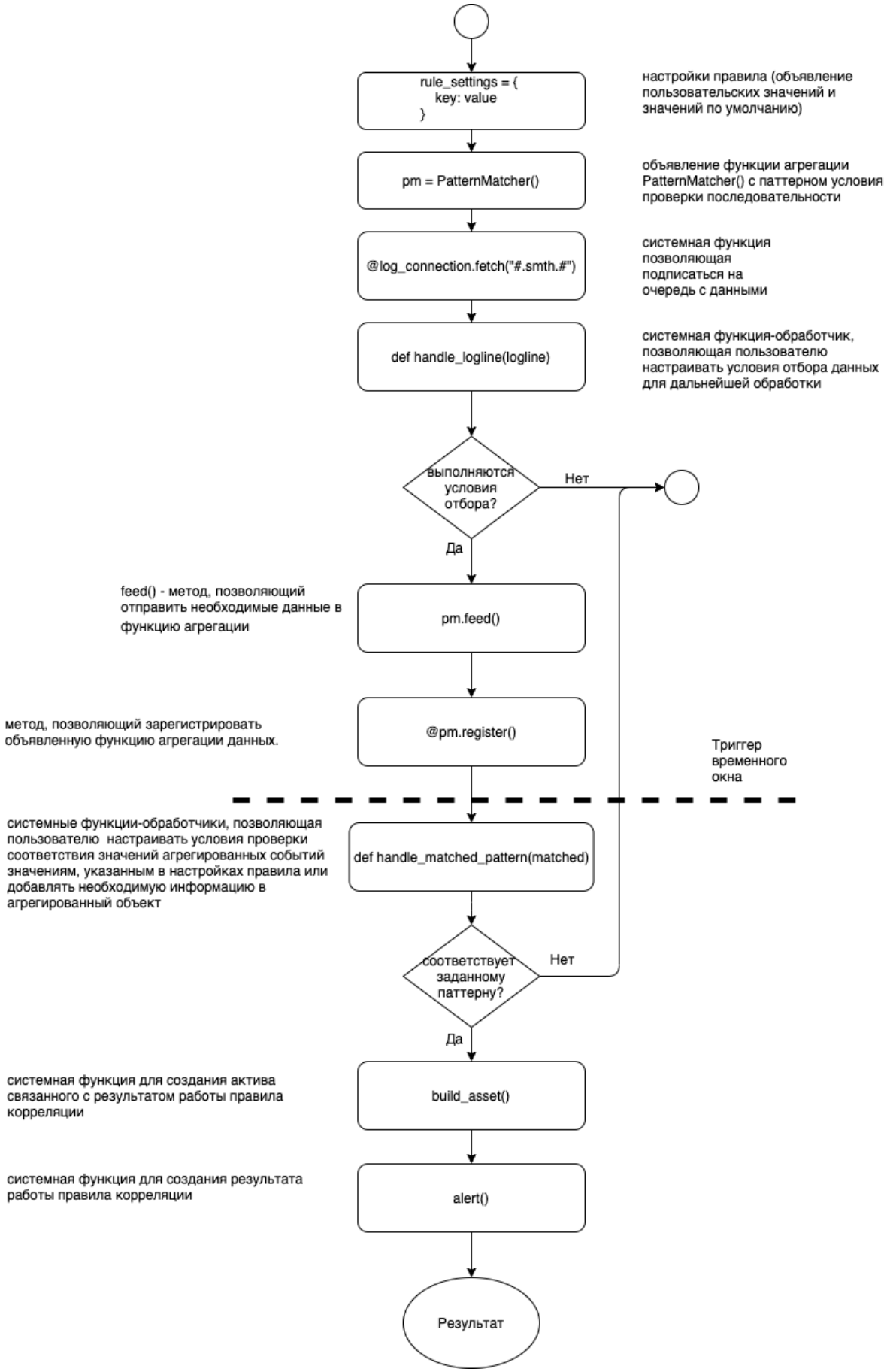

Рисунок 4 - Корреляция событий по шаблону PatternMatcher

Добавление событий в функцию корреляции производится с помощью метода `feed()`:

```
@log_connection.fetch('#.microsoft.windows.os.authentication.#')  # метод для получения событий на корреляцию
def handle_logline(logline):  # функция принимающая полученные события на корреляцию

    if logline.get("observer.event.id") in ['4624', '4672']:
        # Преобразование составного ключа, который должен совпадать у обоих событий
        key_user = logline.generate_key('target.user.id','target.session.id','observer.host.fqdn')

        pattern_matcher.feed(
            event=logline.observer.event.id,  # ключ для передачи в шаблон
            key=key_user,  # ключ группировки (например имя пользователя)
            logline=logline)
```

Удаление событий из функции корреляции производится с помощью метода `cancel()`:

```
pattern_matcher.cancel(
    event=event_logline,  #  удаляемое значение. Если задано None будут удалены все значения для key
    key=event_key)  # ключ группировки
```

Примеры использования регулярных выражений в паттернах:

Передача паттерна в виде массива

```
PatternMatcher('30m', ['login', 'login'], order_by='custom_field')
```

Передача паттерна в виде сложения массивов

```
PatternMatcher('30m', ['login'] + ['login'], order_by='custom_field')
```

Пример необходимости передачи паттерна через сложение массивов:

```
N = 3
codes = {4531: 'failure', 4221: 'success'}
pattern = ['failure'] * N + ['success']
pm = PatternMatcher('30m', pattern, order_by='custom_field')
if code in codes:
      pm.feed(codes[code], user, logline)
```

Функция	**Код**
Совпадение 'failure', что-то и 'success'	`pattern = ['failure', '', 'success']`
Совпадение 'failure', что-то за исключением 'failure' and 'success'	`pattern = ['failure', '(?!failure)', 'success']`
Совпадение 'failure', 'failure' или 'other' and 'success'	`pattern = ['failure', '(failure|other)', 'success']`
Совпадение 'success' followed by anything except 'success'	`pattern = ['success', '(?!success)']`
Пример объявления PatternMatcher с использование регулярного выражения:	`pm = PatternMatcher("1h", pattern, regex=True)`

2.4.4 Корреляция по отсутствию события в цепочке EventTimer

Функция корреляции отсутствия события используется в случае, если требуется обработка ситуации в которой после получения начального события не происходит получение завершающего события в течение заданного временного окна.

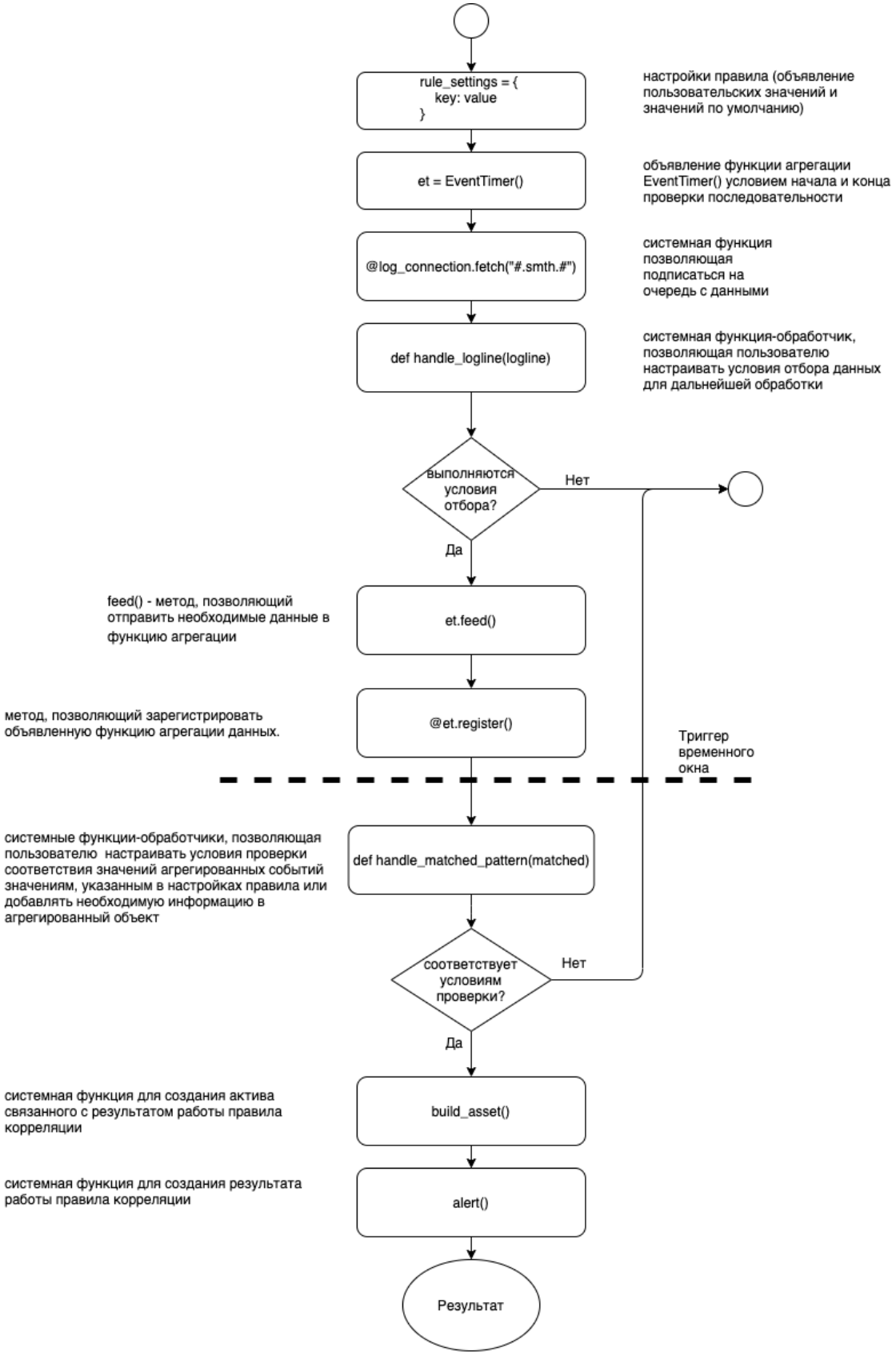

Рисунок 5 - Корреляция по отсутствию события в цепочке EventTimer

Объявление:

```
event_timer = EventTimer(
    rule_settings["detection_windows"],  # размер окна группировки
    start="start",  # ключ начального события
    end="finish",  # ключ конечного события
    interval=30)  # интервал проверки (секунды)
```

Регистрация осуществляется с помощью декоратора `register()`:

```
@event_timer.register()
```

В случае успешной корреляции событий вызывается зарегистрированная функция обработки результатов, которая в качестве параметра получает объект `AggregatedLogline` (часто используется параметр с именем `matched`).

```
@pattern_matcher.register()
def handle_matched_pattern(matched):
    print(matched)
```

Добавление событий в функцию корреляции производится с помощью метода `feed()`:

```
@log_connection.fetch('#.microsoft.windows.os.authentication.#')  # метод для
получения событий на корреляцию
def handle_logline(logline):  # функция принимающая полученные события на
корреляцию
    key = logline.generate_key("ip", "target.threat.name", "path")  # ключ
группировки
    if logline.reaction.executed.name == "detect":
          event_timer.feed("start", key, logline)
    else:
          event_timer.feed("finish", key, logline)
```

Удаление событий из функции корреляции производится с помощью метода `cancel`:

```
event_timer.cancel(
    event=event_logline,  #  удаляемое значение. Если задано None будут удалены
все значения для key
    key=event_key)  # ключ группировки
```

2.5. Использование хранилищ значений и табличных списков

Важной частью разработки правил корреляции является использование постоянных хранилищ и табличных списков.

Хранилища значений чаще всего используются для профилирования правил корреляции к определенным условиям инфраструктуры. К ним можно отнести хранилища, содержащие информацию о сетях, активах критической инфраструктуры, критичные учетные записи и тому подобное.

Содержимое таких хранилищ можно использовать как белый и\или черный список для определенного типа правил корреляции.

Для их создания необходимо в режиме редактирования правила создать новое "Связанное хранилище значений". В поле "Глобальный набор значений" можно объявить следующие типы данных:

- Функции
- Переменные
- Массивы
- Множества

В зависимости от того, что содержится в хранилище, его вызов может осуществляться через:

- Функцию exec (для хранилищ, в которых содержатся функции и переменные)
- Обращение к хранилищу напрямую через функцию stores (для хранилищ, в которых содержатся массивы и множества)

Ниже представлены примеры вызова для каждого из описанных вариантов:

Вызов хранилища с описанием сегментов сети

```
exec(stores["customer_networks"])
```

Хранилище customer_networks:

```
home_net = Networks("192.168.0.0/16")
dmz_net = Networks("172.168.100.0/24")
```

Вызов хранилища с белым списком учетных записей

```
whitelist_username = [user_data[0] for user_data in stores["whitelist"]]
```

Хранилище whitelist:

```
# Имя пользователя, адрес источника, fqdn источника, адрес получателя, fqdn
получателя
# target.user.name, initiator.host.ip, initiator.host.fqdn, target.host.ip,
target.host.fqdn

[
      ['jump_user', ['192.168.0.1'], ['ts01.demo.local'], ['172.164.0.1'],
['vbs.demo.local']],
      ['jump_user2', ['192.168.0.2'], ['ts01.demo.local2'], ['172.164.0.2'],
['vbs.demo.local2']],
      ['jump_user3', ['192.168.0.3'], ['ts01.demo.local3'], ['172.164.0.3'],
['vbs.demo.local3']],
]
```

Пример вызова и использования активного хранилища в правиле представлен ниже: Объявление переменной, содержащей данные из RVS хранилища с информацией об активах assets_info = RVS('assets_info')

Использование переменной, содержащей информацию из хранилища (последнее условие фильтрации):

```
if logline.observer.event.id in {'4624', '4625'} and logline.event.auth.method.id == '10' and logline.target.user.id.startswith('S-1-5-21') and assets_info.get({"ip": logline.get('initiator.host.ip'), "groups": "ДМЗ"}):
```

2.6. Динамические табличные списки

Табличные списки (Rapid Value Store) являются видом активного хранилища (автоматически изменяемого, в зависимости от условий).

Может использоваться для дополнительной фильтрации при работе с обогащением из таких источников как: Active Directory, Активы и т.д. А также для добавления идентификаторов активов и пользователей в "карантин", для исключения повторных сработок до решения инцидента.

Хранилища могут быть созданы вручную и автоматически посредством обработки событий от источника "Kaspersky-SecurityCenter-db-host-activity" и правила "AV_KES_Hosts with old bases and without workable antivirus". А также посредством исполнения скриптов, получающих информацию из Active Directory и активов Платформы.

Для вызова табличных списков в коде правила необходимо использовать функцию RVS.

Для начала нужно определить коннектор к RVS, для этого используем:

`my_collection = RVS(collection_name, index[optional])`

- `collection_name` - str содержащий название коллекции, например "loglines"
- `index` - str содержащий название индекса(обязательного уникального поля), например "id", "ip" (необязательное поле)

Далее используем созданное подключение для управления коллекцией:

`my_collection.get(pattern[optional])` - получить первый документ в коллекции

- pattern - json содержащий условие по которому проводится поиск в формате json, например {"ip":"1.1.1.1"}(необязательное поле)

получить все документы в коллекции - `my_collection.get_all(pattern[optional])`

- `pattern - json` содержащий условие по которому проводится поиск в формате json, например {"ip":"1.1.1.1"}(необязательное поле)

поместить документ в коллекцию - `my_collection.set(document[json])`

- `document` - json содержащий документ, который вы хотите добавить в коллекцию в формате json, например {"ip":"1.1.1.1"}

поместить документ в коллекцию с TTL - `my_collection.set_with_ttl(document[json], ttl[int])`

- `document` - json содержащий документ, который вы хотите добавить в коллекцию в формате json, например {"ip":"1.1.1.1"}

- `ttl` - int количество секунд, через которое документ будет удален из коллекции, например 60

добавить индекс в коллекцию - `my_collection.add_index(index[str])`

- `index` - str содержащий название индекса (обязательного уникального поля), например "id", "ip"

показать все открытые коллекции - my_collection.list()

отчистить коллекцию от документов (сохранив индексацию) - `my_collection.clear(pattern[optional])`

- `pattern - json` содержащий условие по которому проводится отчитстка (ВСЕ поля подходящие под данный фильтр будут удалены) json, например {"ip":"1.1.1.1"}(необязательное поле)

удалить один элемент по фильтру (сохранив индексацию) - `my_collection.remove(pattern)`

- `pattern` - json содержащий условие по которому проводится удаление (ПЕРВЫЙ найденный элемент) json, например {"ip":"1.1.1.1"}

удалить коллекцию - my_collection.drop()

подсчет количества элементов по фильтру (или всчех элементов в коллекции) - `my_collection.count(pattern[optional])` -

- `pattern` - json содержащий фильтр, например {"ip":"1.1.1.1"}

обновить запись по фильтру - my_collection.update(pattern)

- `pattern` - json содержащий фильтр, например {"ip":"1.1.1.1"}

обновить все записи по фильтру - my_collection.update_all(pattern, data)

- `pattern` - json содержащий фильтр, например {"ip":"1.1.1.1"}