Платформа Радар

Описание редактора Lua для правил корреляции

Версия 4.2.3

ООО «Пангео Радар»

Оглавление

1. Структура правила

Пример правила по умолчанию, которое автоматически создается при добавлении в платформу правила:

```
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", "")
  -- alert({
      -- template = template,
      -- risk_level = risk_score,
      -- asset_ip = logline:get_asset_data("target.host.ip"),
      -- asset_hostname = logline:get_asset_data("target.host.hostname"),
      -- asset_fqdn = logline:get_asset_data("target.host.fqdn"),
      -- asset_mac = "",
      -- create_incident = create_incident,
      -- assign_to_customer = assign_to_customer,
      -- logs = {loglines},
      -- trim_logs = 1000, -- макс. количество записей logs для записи в сработку
      -- first_and_last_logs = false, -- передать первую и последнюю запись в logs
      -- meta = meta,
      -- incident_identifier = incident_identifier
  -- })
end

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

Вверху находится блок с переменными, которые отвечают за настройку группера и срабатывание правила:

Строка формата	Пример строки с датой
local detection_windows	Размер окна группировки - временной интервал, в течение которого будет выполняться группировка событий. Формат записи: число со строчным суффиксом. Возможные суффиксы: - ms - миллисекунды; - s - секунды: - m - минуты; - h - часы

Строка формата	Пример строки с датой
local create_incident	Создавать ли инцидент на основании сработки правила. Возможные значения: -true - создавать; - false - не создавать.
local assign_to_customer	Переводить ли инцидент в статус "Назначен" после создания. Возможные значения: - true- статус "Назначен" будет присвоен инциденту после создания; - false - инцидент будет создан в статусе "Новый".
local risk_score	Уровень риска, который будет присвоен при сработке правила. Допустимые значения: 0, 1, 2, 3, 4, 5 ,6, 7, 8, 9, 10.
local grouped_by	Перечень полей нормализованного события, по которым будет выполняться группировка
local aggregated_by	Перечень полей нормализованного события, по которым будет выполняться агрегация
local grouped_time_field	Поле нормализованного события, по которому будет вычисляться время события
local template	Шаблон сообщения инцидента

Примечание: подробнее о настройках группера см. раздел «Групперы».

Функция `on_logline` отвечает за обработку потока событий (далее логлайн). Функция всегда должна быть в правиле, вызывается каждый раз коррелятором при поступлении логлайна, соответствующего фильтрам потока событий, добавленных в правило.

Параметр `logline` позволяет обращаться к текущему логлайну и имеет следующие методы:

Строка формата	Пример строки с датой
logline:**raw**()	Возвращает текст логлайна (строка json)
logline:**ge**t(path, default)	Получить значение поля логлайна по пути (см. **get_field_value**)
logline:**gets**(path, default)	Тоже самое, что и logline:get, с тем отличием, если поле (**path**) не найдено, то возвращается пустая строка (а не **nil**)
logline:**exist**(path)	Проверяет, существует ли указанное (**path**) поле в логлайне (событии)
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 по полям 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.

2. Фильтры

Фильтры отвечают за предфильтрацию логлайнов по правилам описанных в настройке фильтра. По возможности переносите часть условий из правила в фильтр, это позволит более эффективно разбирать поток.

Редактировать фильтры можно как из отдельного раздела (**Веб-интерфейс → Коррелятор → Фильтры потока событий**), так и из редактора правил нажатием на имя фильтра.

3. Макросы

Макросы — это подключаемые модули, которые могут содержать, как и переменные, так и расширять функционал с помощью функций. Импортируется как есть, целиком. Соотв. если в модуле есть определение функции `function test …,` то и использовать ее в правиле следует напрямую **test**().

Редактировать макросы можно как из отдельного раздела (**Веб-интерфейс → Коррелятор → Макросы**), так и прямо из редактора правила нажатием на имя макроса.

4. Группперы

Группер предназначается для аккумуляции событий (как правило, группер насыщается данными в процедуре `on_logline`) с последующей обработкой этих данных, либо проверке событий на соответствие заданному паттерну. Результатом является “сработка” группера, а именно вызов задаваемой функции (`callback`). Периодическая обработка (подсчёт) событий происходит в фоновом режиме с заданным периодом.

Существует два вида группперов, “стандартный” и “pattern matcher”, все функции стандартного так же доступны для pattern matcher, но не наоборот.

Типичный пример использования “стандартного” группера: *требуется подсчитать уникальное количество сетевых адресов за некоторый период (окно), обращающихся к какому-либо ресурсу, и если оно превышает некоторый порог, то создаем инцидент*.

Типичный пример использования “pattern matcher”: *требуется проверить, успешно ли отработал антивирус, то есть был “пойман” вирус, но его лечение или помещение в карантин не произошло в течении 5 минут*.

Определение стандартного группера:

```
grouper1 = grouper.new(
    grouped_by, -- поля группировки
    aggregated_by, -- поля аггрегации
    grouped_time_field, -- поле события где хранится время
    detection_windows, -- окно группера
    on_grouped, -- функция сработки группера (callback)
    -- опциональный параметр с доп. настройками
    {
        group_time="10s", -- время пересчета группера (5 сек по умолчанию, тут меняем
на 10 сек)
        sub_window="5s" -- указываем "меньшее" окно, в рамках которого будет идти
обсчет группера,
                        -- если разница во времени в событиях в группере меньше чем
это окно, то вызова on_grouped не будет
    }
)
```

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
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

Суффикс	Величина времени
s	Секунды
m	Минуты
h	Часы

- `on_grouped` – функция, вызываемая при срабатывании группера. Данная функция в скрипте правила должна объявляться ранее, чем создание группера.
- `grouper1` - объект группера. Может содержать следующие методы:

Метод	Описание
grouper1:**feed**(logline)	“Насыщение” группера. Событие передается для обработки групперу (добавляется в очередь)
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 правила со “стандартным” группером:

```
local detection_windows = "15s" -- окно группера
local create_incident = true -- создать инцидент
local assign_to_customer = false -- назначить инцидент пользователю
local risk_score = 2 -- уровень риска
local grouped_by = {"target.host.ip"} -- группировать по полям
local aggregated_by = {"target.host.ip"} -- аггрегировать по полям
local grouped_time_field = "@timestamp,RFC3339Nano" -- имя поля со временем и
его формат
local template = ["Результат анализа.
С узла {{ .First.initiator.host.ip | join ", " }} была произведена попытка
сканирования"] -- шаблон сообщения инцидента

function on_logline(logline)
  -- здесь возможна дополнительная фильтрация по полям из события
  grouper1:feed(logline) -- "насыщаем" группер
end


function on_grouped(grouped)
    -- отладочное сообщение
    log("agg total: "..grouped.aggregatedData.aggregated.total.." for hash key
"..grouped.key)
```

```
    if grouped.aggregatedData.aggregated.total >= 5 then

        -- использованиеи доп. аггрегации по заданному полю
        resTmp = grouper1:countAgg({"target.ip"})
        check_ok = false
        for k, data in pairs(resTmp) do
            for keyCount, count in pairs(data) do
                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

        logline = grouped.aggregatedData.loglines[1]

        meta = {var = 123}

        alert({
            template = template,
            risk_level = risk_score,
            asset_ip = logline:get_asset_data("target.ip"),
            asset_hostname = logline:get_asset_data("target.hostname"),
            asset_fqdn = logline:get_asset_data("target.fqdn"),
            asset_mac = "",
            create_incident = create_incident,
            assign_to_customer = assign_to_customer,
            logs = grouped.aggregatedData.loglines,
            trim_logs = 100,
            meta = meta,
            incident_identifier = ""
        })

        grouper1:clear() -- очищаем данные текущей сработки
    end
end


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

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

Поле	**Тип**	**Описание**
grouped.key	Строка	“Ключ” группера, собранные в одну строку значения полей группировки
grouped.groupedFields	Массив строк	Массив полей группировки

Поле	Тип	Описание
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	Общее количество событий в группере, для того чтобы получить сумму по всем полям аггрегации, нужно перемножить на кол-во полей (в некотором смысле вводит в заблуждение, вероятно перемножать надо автоматом и сделать отдельное поле для получения количества всех событий в группере)
grouped.aggregatedData.aggregated.countByField	Объект	Поля агрегации и их значения со счетчиками Представление в виде lua объекта: {"agg_field_1": [{"agg_value": count},...], ...} Где agg_field_1 имя поля агрегации, agg_value - значение поля агрегации Пример доступа: count = grouped.aggregatedData.aggregated.countByField['agg_field_1']['agg_value']
grouped.aggregatedData.unique.data	Объект	Уникальные значения полей агрегации Представление в виде lua объекта: {"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 Представление в виде lua объекта: {"agg_field_1": count, ...} Где agg_field_1 имя поля агрегации
grouped.aggregatedData.unique.total	Число	Сумма количества (счетчиков) уникальных значений по каждому полю агрегации
grouped.aggregatedData.unique.valuesByField	Объект	Уникальные значения по парам полей агрегации, может быть полезно, когда нужно получить вложенные уникальные значения по определенному полю. Например: получить уникальные команды выполняемые для определенного типа сообщений

Для доступа к агрегированным данным (ранее `grouped.aggregatedData.aggregated.data`) следует использовать функцию группера **countAgg**.

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"}, -- поля
группировки
    {}, -- поля аггрегации
    {"@timestamp"}, -- поля сортировки
    pattern, -- определение паттерна
    "@timestamp", -- поле содержащее время события
    detection_windows, -- время окна
    on_matched - коллбэк
)
```

Формат записи паттерна:

```
{ field = "имя поля", values = массив_значений [, (опционально) count =
счетчик_повторов] [, (опционально) absent = true], [, (опционально) exact =
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.
- `count` - количество повторов значений (равно или больше - по умолчанию) для определения соответствия паттерну, если `count` не указывается (и это не absent), то он будет равен 1;
- `exact` - флаг, указывающий на то, что значений должно быть точно равно счетчику повторов (изменение поведения по умолчанию у `count`).

Функция коллбэк отличается от стандартного группера:

```
function on_matched(grouped, matchedData)
    -- отладочное сообщение
    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`.

Про работу паттерн матчера см. раздел «Паттерн матчер».

4.1. Паттерн матчер

Паттерн матчер работает в следующих режимах

- Обычный;
- С отсутствующим(и) элементом в начале (`absent@begin`);
- С отсутствующим(и) элементом в конце (`absent@end`).

4.1.1 Обычный режим работы

Пример паттерна:

```
pattern = {
    { field = "observer.event.id", values = {"4720"}, count = 1 },
    { field = "observer.event.id", values = {"4726"}, count = 1 },
}
```

Данный паттерн означает, что в потоке (поступающие в групер события, функция группера **feed**) будут проверяться события идущие в порядке (друг за другом) со значениями в поле `observer.event.id` *4720* и *4726*, в количестве от 1, т.е. последовательность *4720*, *4720*, *4720*, *4726*, *4726* вызовет только 1 сработку (матч), если требуется проверка точного количества (повторов) элементов, то требуется указать флаг **exact** в паттерне.

Пример паттерна с **exact**:

```
pattern = {
    { field = "observer.event.id", values = {"4720"}, count = 1, exact = true },
    { field = "observer.event.id", values = {"4726"}, count = 1 },
}
```

при таком паттерне последовательность *4720*, *4720*, *4720*, *4726*, *4726* вызовет тоже 1 сработку (матч), но в нее не попадут первые два события (*4720*, *4720*), т.к. условием является одно значение *4720* и одно или больше значений *4726*.

Обычный паттерн так же, может содержать в себе absent в середине паттерна, например:

```
pattern = {
    { field = "observer.event.id", values = {"4720"}, count = 1 },
    { field = "observer.event.id", values = {"4721"}, absent = true },
    { field = "observer.event.id", values = {"4726"}, count = 1 },
}
```

При таком паттерне, сработка(матч) будет для последовательностей *4720*, *4726*. Но, если в середине окажется значение *4721 (4720, 4721, 4726)*, то сработки не произойдет.

4.1.2 Режим работы Absent@Begin

Пример паттерна:

```
pattern = {
    { field = "action", values = {"av1"}, absent = true },
    { field = "action", values = {"v1"}, count = 1 },
```

```
    { field = "action", values = {"v2"}, count = 1 }
}
```

Данный паттерн означает, что для сработки (матча) первый элемент (значение *av1* в поле **action**) при сопоставлении должен отсутствовать перед последовательностью *v1, v2*. (в последовательности **count**, **exact**, **absent** в “середине”, ведут себя так же, как и в обычном паттерне)

4.1.3 Режим работы Absent@End

Пример паттерна:

```
pattern = {
    { field = "action", values = {"v1"}, count = 1 },
    { field = "action", values = {"v2"}, count = 1 },
    { field = "action", values = {"av1"}, absent = true }

}
```

Данный паттерн означает, что для сработки (матча) последний элемент (значение *av1* в поле **action**) при сопоставлении должен отсутствовать после последовательностью *v1, v2*. (в последовательности **count, exact, absent** в “середине”, ведут себя так же, как и в обычном паттерне).

Внимание: *сопоставление будет происходить при достижении времени жизни события (окно групера). т.е. (из примера) проверка на отсутствие av1 произойдет, когда v1 будет удаляться из данных группера*.

5. Массивы

Примеры использования массивов:

- `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**`(массив_булевских_значений)` – вернет `true`, если хоть один из элементов массива = `true`;
- **contains**`(массив, значение [, тип_сравнения])` – возвращает `true`, если хоть один элемент массива подходит к значению.

  Тип сравнения (опциональный флаг, строка, по умолчанию "exact"):

  - **"", "exact"** - сравнивается как есть 1 к 1
  - **"prefix"** - сравнивается по началу
  - **"suffix"** - сравнивается по окончанию
  - **"sub"** - сравнивается по подстроке

  Пример:

  ```
  if not contains({"one", "two"}, "on", "sub") then
      error("contains(sub) failed")
  end
  ```

6. Функции

6.1. 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?)`	Поиск по Regexp, возвращает true, если совпадение найдено
`("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

6.1.1 Альтернативные функции работы со строками

Дублируют функции вида `("строка"):функция`, с тем отличием, что принимают строку на вход в качестве первого параметра, если же вместо строки передается `nil`, то он считается пустой строкой. Использование может быть полезно в купе с функциями **logline:get**, когда самое поле отсутствует (для этого безопаснее использовать **logline:gets**, например: `logline:gets("non.existent.field"):trim()`,
альтернатива: `str_trim(logline:get("non.existent.field"))`.

Список функций:

- str_len("строка");
- str_sub("строка", начало);
- str_trim("строка");
- str_split("строка", "разделитель");
- str_search("строка", "поиск");
- str_endswith("строка", "подстрока");

- str_startswith("строка", "подстрока");
- str_upper("строка");
- str_lower("строка").

6.2. Работа с логлайнами (json в строке)

- **get_field_value**(источник, “путь”) – получить значение в пути из источника. Источник либо логлайн, либо json в строке.

  Путь - ссылка на поле json, например target.ip, будет соответствовать {“target”: {“ip” : “значение“}}

  Более подробно см. GitHub - tidwall/gjson: Ge... @GitHub

- **get_fields_value**(источник, массив_путей[, значения по умолчанию]) – возвращает массив значений из источника. Источник либо логлайн, либо json в строке.

  Пример:

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

  Пример с значениями по умолчанию:

  ```
  vals = get_fields_value(logline, {"number_non_ex", "string_non_ex", "non_ex", "wo_def"}, {-1, nil, ""})
  -- значения после выполнения: vals[1] = -1, vals[2] = "", vals[3] = ""
  -- примечания: дефолтные значения nil будут изменены на пустую строку ""
  -- (lua tables плохо работают с nil значениями)
  ```

- **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).*

- **new_logline**(параметр) – создать новый объект типа logline (как параметр logline в функции on_logline).

  Параметром на вход (аргумент функции) может быть строка (в формате json) или таблица.

  Пример:

  ```
  js = [[{"field": "value"}]] -- пример логлайна (json строка)
  ll = new_logline(js) -- создаем логлайн
  obj = ll:decode() -- декодируем в объект (таблицу)
  obj.field = "value1" -- меняем поле
  log(new_logline(obj):raw()) -- кодируем снова в логлайн и отображаем результат
  ```

6.3. 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
```

6.4. 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.

Разница между параметром “ключ” и “поле ключ” в следующем: есть "ключ" назовем его "общий ключ" или ID записи, а есть "поле ключ", их может быть много. Общий ключ — это конкатенация значений всех полей ключей. А "ключ" необходимо рассматривать как ID записи, т.е. чтобы добраться до полей, нужно знать этот ID и именно по нему идёт обращение.

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

Имя метода	Описание
	время + указанное TTL).
test_storage:**get**(“ключ“, “имя_колонки”, “опциональное_значение_по_умолчанию“)	Возвращает значение по ключу для указанной колонки. Если запись не найдена и не указано значение по умолчанию, будет возвращен nil.
test_storage:**gets**(“ключ“, “имя_колонки”, “опциональное_значение_по_умолчанию“)	Тоже самое что и test_storage:**get**, за исключением того, что если запись не найдена и не указано значение по умолчанию, будет возвращена пустая строка (а не nil).
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.
	Пример:
	`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", "значение")	Ищет заданное значение в колонке с указанном именем во всем справочнике, возвращает имя первого ключа, если значение нашлось, иначе nil. Формат “значение” см. ниже.
test_storage:**searchs**("имя_колонки", "значение")	Тоже самое что и test_storage:**search**, за исключением того, что если значение не нашлось, то вернется пустая строка (а не 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** Поддерживаются регулярные выражения, для их использования следует указать префикс ~/ для строки поиска. Например: ~/substr Только **search_all**: для получения списка ключей с помощью сравнения колонок со значениями типа число можно использовать операторы сравнения < > <= >=, Пример: test_storage:search_all("number_column", ">1000") -- получить список ключей у которых значения в колонке number_column превышают 1000
test_storage:**math_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). Список_ключей - массив из ключей (массив строк идентификаторов строк табличного списка), его возвращает, например, **search_all**. Пример {“id1”, “id2”} Если список ключей пуст или nil, то подсчет будет произведен для всего справочника
test_storage:**math_count**(список_ключей, "имя_колонки")	Выполняет калькуляцию по указанным ключам по указанной колонке. Возвращает количество записей в табличном списке по указанной колонке.

Имя метода	Описание
	Если список ключей пуст или nil, то подсчет будет произведен для всего справочника.
test_storage:**math_min**(список_ключей, "имя_колонки")	Выполняет калькуляцию по указанным ключам по указанной колонке. Возвращает минимальное значение в табличном списке по указанной колонке. Если список ключей пуст или nil, то подсчет будет произведен для всего справочника.
test_storage:**math_max**(список_ключей, "имя_колонки")	Выполняет калькуляцию по указанным ключам по указанной колонке. Возвращает максимальное значение в табличном списке по указанной колонке. Если список ключей пуст или nil, то подсчет будет произведен для всего справочника.
test_storage:**math_avg**(список_ключей, "имя_колонки")	Выполняет калькуляцию по указанным ключам по указанной колонке. Возвращает среднее значение в табличном списке по указанной колонке. Если список ключей пуст или nil, то подсчет будет произведен для всего справочника.
test_storage:**math_sum**(список_ключей, "имя_колонки")	Выполняет калькуляцию по указанным ключам по указанной колонке. Возвращает суммарное значение в табличном списке по указанной колонке. Если список ключей пуст или nil, то подсчет будет произведен для всего справочника.
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"}))`

Примечание: *если параметр* `имя_колонки` *- пустое, то используется имя по умолчанию (“value”).*

6.5. 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)

6.6. 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

6.7. Вспомогательные функции

В качестве вспомогательных функций могут быть использованы следующие методы:

- **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-адрес 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.

  Пример:

  ```
  event_register({
    finding_title = "Антивирус - Обнаружено вредоносное ПО",
    asset_type = "Host",
    logline_summary = { { test = 1 }, { test = 2 } }, -- array of tables
    result_asset_fqdn = "matched_logline.target.host.fqdn",
    result_description = "",
    result_created_at = "matched_logline.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.collector.timestamp",
    rule_name = "AV-001-Malware detected and not removed Users",
    result_occurred_at = "matched_logline.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 = 0.5,
    result_solution = "",
    result_analysis_output = ""
  }, {
    "rule_id"

  })
  ```

- **now_in_ms**() – возвращает локальное текущее время в миллисекундах (*UnixMilli*);
- **type**(значение) – возвращает тип значения:
  - ”bool” - булевское значение

- ”number” - число
- ”string” - строка
- ”nil” - пустое
- ”function” - функция
- ”table” - массив
- ”user” - внутренний объект

- **uuid()** – возвращает сгенерированный UUID (строка);
- **compare**(left, “comparator”, “right”) – функция сравнения, сделана для возможности сравнивать разные типы, пример:

  ```
  compare(123, "==", "123") -- вернет true
  compare(true, "==", "1") -- вернет true
  ```

  “comparator” является одним из доступных операторов сравнения:

  “==“, “!=“, “<“, “<=“, “>“, “>=“, ”<“, “>”

  Если функция не сможет привести типы, которые возможно сравнить, всегда вернет `false`.

6.8. Алерт

Поле	Тип	Описание

template	string	Шаблон инцидента. Формирование происходит согласно goland text template . Из этого шаблона формируется результат анализа инцидента.
		
		Возможные атрибуты:
		- .First - первый логлайн
		- .Last - последний логлайн
		- .Loglines - массив логлайнов
		- .Meta - опциональные параметры переданные из правила (далее эти данные сохраняются как дополнительные поля)
		- .Vars - опциональные параметры (эти данные нигде не сохраняются)
		- .Grouped.aggregatedData.aggregated - обращение к данным агрегации (*aggregated.total*, *aggregated.count*)
		- .Grouped.aggregatedData.unique - обращение к уникальным данным агрегации (*unique.total*, *unique.data*, *unique.count*, см. пример ниже)
		
		Пример шаблона:
		
		`Действие {{ .First.action }}", инициатор {{ .First.initiator.ip }}.`
		`Мета: {{ .Meta.var }}`
		`Arr test: {{ .First.target.arr | join ", " }}`
		`Перечень адресов:`
		`{{ range $index, $element := .Loglines }}{{ println "-" .target.ip "порт" .target.port }}{{ end }}`
		
		`Количество уникальных адресов {{ index .Grouped.aggregatedData.unique.count "target.ip" }}`
		`Список уникальных адресов:`
		`{{ range $index, $element := index .Grouped.aggregatedData.unique.data "target.ip" }}`

Поле	Тип	Описание

		`- {{ print $element }}{{ end }}`
		
		`Минимальное значение: {{ .Vars.ints | min }}`

risk_level	float	Уровень риска (от 0 до 10)
asset_ip	string	Значение IP актива, для заполнения рекомендуется использовать функцию logline:**get_asset_data**, Например: logline:**get_asset_data**("observer.host.ip"). Это же актуально и для *asset_hostname, asset_fqdn, asset_mac*
asset_hostname	string	Значение Hostname актива
asset_fqdn	string	Значение FQDN актива
asset_mac	string	Значение MAC актива
create_incident	bool	Флаг создавать инцидент
assign_to_customer	bool	Флаг назначить пользователю
incident_group	string	Опциональный параметр. Группа инцидентов (имя!(title) не ID)
logs	array of strings/loglines	Массив логлайнов который будет записан вместе с результатом в БД
trim_logs	int	Опциональный параметр. Обрезать массив логлайнов согласно этому ограничению
first_and_last_logs	bool	Опциональный параметр. Отправлять только первый и последний логлайн
meta	object	Опциональный параметр. Отправить и записать в БД опциональные параметры (сохраняются в т.ч. как дополнительные поля у происшествия)
vars	object	Опциональный параметр для доступа из шаблона (не сохраняются в БД)
incident_identifier	string	Опциональный параметр. Идентификатор инцидента
mitre	array of strings	Опциональный параметр. Список техник Mitre. Например: mitre={"T1585", "T1585.001"}

Пример:

```
alert({
    template = "Обнаружено изменение временного атрибута у файла с помощью утилиты
"touch". С хоста IP: {{ .First.initiator.host.ip | join ", " }}", Пользователем: {{
.First.initiator.user.name | join ", " }}"",
    risk_level = 0.5,
    asset_ip = "127.0.0.1",
    asset_hostname = "localhost",
    asset_fqdn = "localhost.pgr.local",
    asset_mac = "12:12:12:12:12:12",
    create_incident = true,
    assign_to_customer = false,
    logs = grouped.aggregatedData.loglines
})
```