Платформа Радар

Описание специальных функций корреляции

Версия 3.6.0

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 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.

1.2. Групперы

Существует два вида групперов: 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
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
RFC822	02 Jan 06 15:04 MST
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.

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

> 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

        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

Поле	Тип	Описание
`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` (массив*булевских*значений)	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

Имя функции	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

1.4.2. Работа с логлайнами (json в строке)

Имя функции	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"})`

Имя функции	Описание
`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

Имя функции	Описание
`sleep(миллисекунды)`	“Засыпает” на указанное количество миллисекунд
`log(значение)`	Выводит значение в лог сервиса. Значением может быть строка, число, объект или булевый тип
`set_debug_value(имя, значение)`	Устанавливает значение отладочной переменной, выводится в результатах тестирования
`error(строка)`	Вызвать ошибку в правиле с описанием “строка”

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`

Имя метода	Описание
`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

Имя метода	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

Имя метода	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

Имя метода	Описание
`is_home_net(строка_с_ip_адресом)`	Проверяет входит ли ip адрес в домашнюю сеть, подсети задаются в конфигурации сервиса logmule

Имя метода	Описание
`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)