Настройка фрод мониторинга: быстрый старт.

По вопросам обращайтесь по адресу: support@cybertonica.ru.

Данное руководство предназначено для пользователей, которые хотят начать писать правила в системе Cybertonica. Также оно может использоваться как справочное пособие.

Предполагается, что читатель обладает базовыми знаниями программирования на уровне понимания что такое циклы, функции, массивы и словари.

Введение

Если вам нужно освежить знания Lua - рекомендуем: https://learnxinyminutes.com/docs/ru-ru/lua-ru/

При написании кода советуем придерживаться следующих принципов: - Избегайте использования глобальных переменных - Давайте переменным понятные имена, код читают чаще чем пишут - Избегайте дублирования кода, оно ведет к ошибкам в будущем - Проверяйте входные данные, им нельзя доверять - Разбивайте код на функции, код функции должен умещаться на одном экране - Разбивайте код на модули - Не забывайте использовать `pcall` в Lua для обработки ошибок. - Помните, о том что работа происходит в распределенной многопоточной среде, не забывайте про гонки (race conditions)

Принципы фрод-скоринга - Работа скоринга происходит на потоке событий, который в теории не имеет начала и конца, поэтому используйте скользящие окна в расчетах. - Общая схема работы: получение входнях данных, опрос поставщиков данных, применение моделей машинного обучения, применение правил - счетчики по временным окнам - множества по временным окнам - TODO: графовые признаки - сравнение признаков за разные периоды - адаптивность правил - размеры выборки и статистическая значимость правил

Структуры данных для вычисления признаков (data structures / feature engineering)

Входящее событие (Event)

В новом протоколе “событие” включает в себя как создание некоторого факта (createEvent старого протокола), так и обновление статуса. Различать планируется по полю `channel`. Для обновлений каналы назвать `update_{channel}`.

События состоит из 4 групп полей: 1) общие поля - идентификатор события, время `t_ms`, `session_id`, `chanenl`, `sub_channel`. 2) поля входящего запроса, `request` 3) поля ответа: `action, score, queues, tags, comments`. Они могут пригодиться, при обработке предыдущего уже проскоренного события. 4) технические поля (tx_id, hostname, timings_mcs, version) - эти поля хотя и доступны в Lua, но бизнес-логике не нужны.

Структура события из ядра системы:

```
pub struct EventRecord {
    /// uuid
    pub tx_id: String,
    /// unix time milliseconds
    pub t_ms: u64,

    ///       ,       "scoring"
    ///           event log
    pub log_type: String,
    ///
    pub team: String,
    ///
    pub version: u16,
    ///              AS IS

    pub session_id: Option<String>,
    pub user_id: Option<String>,
    pub channel: String,
    pub sub_channel: String,
```

```
    pub prev_event_id: Option<String>,
    pub prev_event: Option<Box<EventRecord>>, // TODO: do not write to historyDB to avoid infinite link

    // todo: data_providers: binbase, geoip, fixer, session
    pub session: Option<SessionPublic>,

    pub ip_score: Option<ScoreIpPublic>,

    pub action: String,
    pub score: f32,
    pub queues: Vec<String>,
    pub tags: Vec<String>,
    pub comments: Vec<String>,

    pub error: Option<String>,
    //                       :
    ///                        -
    pub timings_mks: Vec<(String, u64)>,
    ///       , /etc/hostname
    pub hostname: String,
    ///
    pub seq_id: u64,
}
```

Результат скоринга (ScoringResult)

Эту структуру данных должен возвращать обработчик Lua `on_event`.

```
pub struct ScoringResponse {
    pub action: String,
    pub score: f32, // todo: changed from int 0..1000!
    pub queues: Vec<String>,
    pub tags: Vec<String>,
    pub comments: Vec<String>,
    pub ip_score: Option<ScoreIpPublic>
    pub extra: serde_json::Value
}

pub struct ScoreIpPublic {
    pub ip: String,
    pub geo_city_name: String,
    pub geo_country_code: String,
    pub geo_iso_code: String,
    pub geo_latitude: f64,
    pub geo_longitude: f64,

    pub blacklisted: bool,
    #[serde(default)]
    pub tags: Vec<String>,
}
```

- `action` - одно из значений `ALLOW, CHALLENGE, DENY`.
- `score` - число от 0 до 1. Замечание: в ранних версих протокола было от 0 до 1000.
- `ip_score` - результат скоринга ip-адреса, доступен только если в createEvent было передано поле `ip`.
- `extra` - произвольная таблица, которая позволяет расширять скоринг.

Работать с `ScoringResponse` удобнее через объектно-ориентированную обертку, см. utils.lua.

Временные ряды timeseries

Набор счетчиков по времени.

```
add(key, period, t_ms, value)
```

```
key: String, period: String, t_ms: u64, value: f32
period: {number}{unit}, etc. 10m, 1h, 7d
```

Внутренний формат хранения: набор пар время-счетчик, при этом счетчик отвечает за суммирование за период (, +period)

```
sum(key, t_ms, period)
```

Возвращает 0, если ключ не существует.

Множества элементов ограниченное по времени (time sets)

Набор уникаольных строк по времени.

```
add(key: String, period: String, t_ms: u64, value: String)
```

```
nunique(key, t_ms, period) -> int
```

Возвращает 0, если ключ не существует.

```
item(key, t_ms, period) -> [[times], [values]]
```

Возвращает элементы вместе с временными метками. Временные метки - unix time в миллисекундах. Хранение внутри *в секундах*.

Функции работы с проверочными списками (check_lists)

Проверочные списки вида ключ-значение.

Проверка наличия ключа в списке.

```
check_lists:contains(list_name, key)
```

Возвращает `bool`. Если key `nil`, то возвращает `false`. В случае истекшего элемента возвращает `false`. В случае несуществующего списка вызывает ошибку.

Получение записи из списка.

```
check_lists:get(name, key)
```

Возвращает `nil` или таблицу вида:

```
key: str
value: str
comment: str
expires_at:         unix time   ms
created_at:
```

Если key `nil`, то возвращает `nil`.

Добавление элемента в список.

```
check_lists:add(name, key, value, current_time_ms, ttl_period)
```

В случае вставки в несуществующий список возвращает ошибку.

todo: в случае больших списков или распределенной инсталяции эта функция имеет специфику!

Хранилище таблиц kvstorage

Хранилище произвольных таблиц.

Получение элементов

```
mget
Type:
table -> table
[str] -> [Optional[table]]
```

Сохранение элементов

```
mset
Type:
table, table -> nil
[str], [table] -> nil
```

Поставщики данных

Геолокация IP-адреса (GeoIP)

```
tools:geoip(ip)
string or nil -> table or nil
```

Пример:

```
[(postal_code => 127976),(country_name => Russia),(country_code => RU),(country_subdivision_iso => MOW)
```

Справочник по карточным Bank Identification Code (BIN)

```
tools:bininfo(b)
string or nil -> table or nil
```

Сессионный мониторинг

В объектах `EventRecord` есть поле `session`. Оно может быть либо пустым, либо содержать информацию о сессии с идентификатором `session_id`.

Структура записи `SessionPublic`:

```
pub struct SessionPublic {
    ///            , GUID
    pub session_id: String,
    ///               (     ,                           )
    pub device_id: String,
    ///            ,        unix time (UTC).
    pub start_t_ms: u64,
    ///           ,        unix time (UTC),
    ///                                    ,
    ///
    pub end_t_ms: u64,
    ///       fingerprint            ,
    pub n_fingerprints: u16,
    ///       tracking             ,
    pub n_tracks: u32,
    ///                fingerprint
    pub last_fingerprint_t_ms: u64,
    ///                        tracking
    pub last_track_t_ms: u64,
    /// UserAgent
    pub user_agent: String,     // TODO:               -          ?
    ///                     - web, ios, android
    pub platform: String,
    ///         (             )
    pub alerts: Vec<SessionAlert>,  // TODO: public alerts?
    ///     IP     ,
    pub ips: Vec<String>,
    // TODO: geoip?
    ///     ,
    pub referrers: Vec<String>,
}
```

Структура `SessionAlert`:

```
pub struct SessionAlert {
    ///             , unix time milliseconds.
```

```
    pub t_ms: u64,
    ///             (         tid)
    pub session_id: String,
    ///       ,     , BOT, SUSPICIOUS_IP, COPY_PASTE, DEV_TOOLS, ATO,...
    pub name: SessionAlertName,
    ///
    pub message: String,
    ///            (confidence score            ),       0   1
    pub confidence: f32,
}
```

Актуальная информация содержится в документе “OmniReact сессионный мониторинг. Руководство по API”.