Cybertonica OmniReact. Руководство по интеграции транзакционного скоринга

Version 2.9

email: support@cybertonica.ru

Введение

Cybertonica FraudSuite360TM - это платформа, которая помогает финансовым и платежным организациям бороться с мошенничеством в области онлайн платежей и оптимизировать конверсию.

Система анализирует потоки событий (например, заказы, платежи или входы в мобильный банк) в реальном времени и принимает решения относительно дальнейшей обработки событий. Это решение является результатом анализа различных источников информации, включая данные транзакции, связанные с оплатой, информацию о биометрических сигналах с устройства (сигнатуры устройства) и данные о поведении пользователей, осуществляющих платеж. Другими словами, решается задача транзакционного мониторинга и адаптивной аутентификации на основе анализа риска в режиме реального времени.

Ниже показана схема взаимодействия при скоринге платежа совместно с сессионным мониторингом.

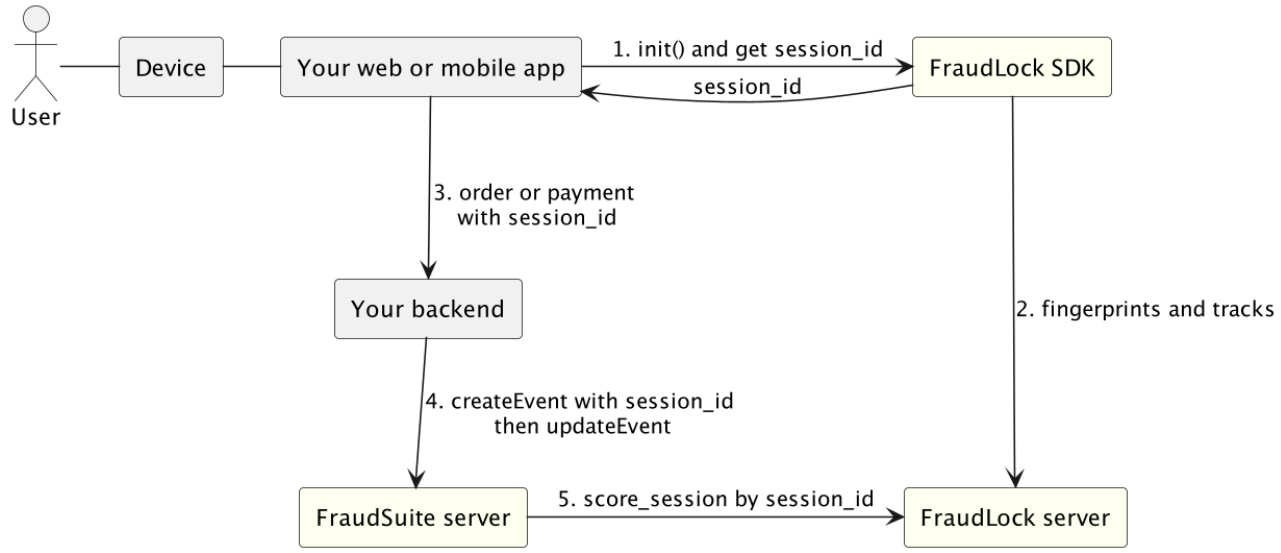

Figure 1: Components

Транзакционный мониторинг может работать совместно с сессионным мониторином. Схема работы: 1. На клиентском приложении генерируется идентификатор сессии и передается вместе с транзакцией. 2. Транзакция вместе с идентификатором сессии отправляется на проверку. 3. Транзакционный мониторинг запрашивает информацию по сессии, включая индикаторы угроз.

Подробности интеграции сессионного мониторинга смотрите в документах: - Руководство по интеграции модуля безопасности. - Руководство по API сессионного мониторинга.

Порядок интеграции

Чтобы интегрироваться с системой, необходимы следующие шаги:

1. Внедрите систему сессионного мониторинга FraudLock360 (опционально).
2. Реализуйте протокол, описанный в этом документе.
3. Получите учётные данные для доступа к тестовой среде.
4. Запустите интеграционные тесты, см. приложение №2.
5. Укажите IP-адреса, с которых будет осуществляться доступ к рабочей среде.
6. Получите учётные данные для доступа к промышленной среде.

Протокол скоринга

Общие принципы

Идентификация событий происходит по идентификатору заказчика `extid` (external ID). Пара `(channel, extid)` должна быть уникальной. При повторной отправке пары будет возвращена ошибка.

Взаимодействие производится по протоколу HTTPS по схеме запрос-ответ, транспортный формат JSON.

Внимание! Настоятельно рекомендуется использовать пул соединений, чтобы избежать длительной установки SSL-соединения при каждом запросе.

Для обеспеспечения возможности обновременной обработки разные потоков событий и обеспечения целостности данных в системе используется механизм *каналов*.

Канал – это поток событий с одинаковой структурой (схемой) полей и с уникальными идентификаторами (extid) в рамках данного потока.

Например, мы можете одновременно проводить мониторинг канала `payment` с карточными платежами и `login` с событиями входа в приложение мобильного банка.

Система позволяет создавать новые каналы и гибко задавать поля данных. Однако, мы рекомендуем придерживаться списка стандартных полей (см. ниже), это позволяет проще переносить правила и модели между разными инсталляциями системы.ы

Также в протоколе предусмотрены *подканалы*.

Подканал – это логическая метка потока событий в рамках определенного канала. Метка проставляется на стороне заказчика, например, этом может быть регион, сегмент клиентов, имя ИТ системы-источника (интернет банк, мобильный банк, и т.д.). Если подканала нет, то используется подканал по-умолчанию с именем `Default`.

Аутентификация запросов

Доступ к API Cybertonica в промышленной среде должен проводиться только с согласованных наборов IP-адресов, остальные адреса должны быть заблокированы на сетевом экране.

Каждый запрос должен содержать заголовки HTTP `X-AF-Team` и имитовставку `X-AF-Signature`.

Значение `X-AF-Signature` вычисляется с использованием алгоритма HMAC-SHA1: `base64(hmac-sha1(secret_key, json_message))`, где `secret_key` - ваш секретный ключ, соответствующий `X-AF-Team`.

Пример кода Python3 для вычисления значения X-AF-Signature.

```
import hmac
import base64
import hashlib
key, json_str = "secret", "...JSON of the message..."
hashed_data = hmac.new(key.encode(), json_str.encode(), hashlib.sha1).digest() b = base64.encodebytes(h
print(b)
```

Тестовые вектора:

JSON	key	signature
`{"msg": "your JSON"}`	`"your secret"`	dvyBUORn7Vtfs3UI7BrESQX1hqM=
`{"msg": "you JSON"}`	`"another secret"`	4xdTopyfav2xiUfkpLxsgJBGz/Y=
`""` (empty string)	`"your secret"`	MZQhUxUjdO9Jnu3hgO1lMnNmwkE=

Методы API

Создание события (createEvent)

Создает новое событие и проводит его проверку.

POST URL: `https://{customer}.cybertonica.ru:7499/api/v2.2/events/{channel}?subChannel={subChannel}`

Замечание: при внедрении периметра заказчика адрес может отличаться.

Запрос:

- `channel`: (обязательно) имя канала, например, `payment, login, order`.

- `subChannel`: (опционально) имя подканала, `Default` по-умолчанию.
- `ip`: (опционально) адрес хоста совершившего транзакцию, при указании совершает проверку адреса.

Тело запроса: JSON с полями, соответствующими `channel`.

Ответ:

JSON с следующими полями:

`id: string`: (для обратной совместимости) Идентификатор события в системе Cybertonica

`channel: string`: (для обратной совместимости) имя канала

`score: int`: Числовое значения оценки риска события: 0 (ноль) - минимальный риск, 1000 (тыс.) - максимальный риск.

`action: string` - рекомендация по дальнейшей обработке события

Значение	Пояснение
ALLOW	операция может быть проведена без дополнительной проверки
CHALLENGE	операция должна быть под- вергнута дополнительной проверке, например, 3DSecure / OTP
DENY	Рекомендуется отклонить операцию

`comments: array[string]` - массив комментариев от правил

`tags: array[string]` - массив тэгов, проставленных правилами

`rules: array[string]` (для обратной совместимости)

`ip_score: object` - данные оценки ip-адреса (если был указан в запросе)

Поле	Пояснение
ip: string	Искомый адрес
geo_city_name: string	Название города размещения искомого адреса
geo_country_code: string	Название страны размещения искомого адреса
geo_iso_code: int	ISO 3166-1 Numeric code код страны размещения искомого адреса
geo_latitude: double	Широта размещения искомого адреса
geo_longitude: double	Долгота размещения искомого адреса
blacklisted: boolean	Наличие адреса в черном списке
tags: array[string]	Потенциальные угрозы от искомого адреса

Возможные теги:

- Drop - спам или просто нежелательный адрес;
- Anonymizer - любые средства анонимизации (прокси/впн/тор);
- Webserver - адрес принадлежит диапазонам облачных провайдеров.

`extra: object` - (опциональное) произвольные данные

Возможные ошибки:

`400 Bad Request`: Неверный запрос.

`401 Unauthorized`: Некорректная подпись. Проверьте алгоритм `X-AF-Team` и `X-AF-Signature`.

`504 Gateway timeout`: Сервер Cybertonica в настоящее время недоступен

Пример:

Запрос:

```
{
    "amount": 100000,
    "currency": 643,
    "bin": "553691",
    "channel": "payment",
    "client_id": "eczb...",
```

```
    "email": "go...@domain.com",
    "exp": 2,
    "extid": "1234-5678-9999",
    "ip": "85.11.11.111",
    "merch_id": "merch....",
    "parent": "CARD",
    "parent_id": "token...",
    "phone_id": "........",
    "sub_channel": "card_merch",
    "t": 1663863862894,
    "session_id": "1663857082a2f-483f-fdbf-1299c928...."
}
```


Ответ:


```
{
  "id": "1663864863dd547a2d-bd98-415d-a393-c278b8059ce4",
  "channel": "payment",
  "action": "ALLOW",
  "score": 10,
  "queues": [
    "MULTIPLE_ACCOUNTS",
    "BOT",
    "SIMILAR_EMAIL",
    "FOREIGN_IP"
  ],
  "tags": [
    "MULTIPLE_ACCOUNTS",
    "BOT",
    "SIMILAR_EMAIL",
    "FOREIGN_IP",
    "TX_DUBLICATE"
  ],
  "comments": [
    "rule comment 1",
    "rule comment 2",
    "Duplicate tx id: ..."
  ],
  "extra": {
    "features": {
      "cid:card:long": 1,
      "cid:card:medium": 1,
      "cid:card:short": 0,
      "cid:card:vlong": 1,
      "cid:device:long": 1,
      "cid:device:medium": 1,
      "cid:device:short": 1,
      "cid:device:vlong": 1,
      "cid:email:long": 1,
      "cid:email:medium": 1,
      "cid:email:short": 1,
      "cid:email:vlong": 1,
      "cid:foreign_ip:long": 0,
      "cid:foreign_ip:medium": 0,
      "cid:foreign_ip:short": 0,
      "cid:foreign_ip:vlong": 0,
      "cid:parent:long": 1,
      "cid:parent:medium": 1,
      "cid:parent:short": 1,
      "cid:parent:vlong": 1,
      "cid:parent_selected:long": 0,
      "cid:parent_selected:medium": 0,
```

```
"cid:parent_selected:short": 0,
"cid:parent_selected:vlong": 0,
"cid:phone:long": 2,
"cid:phone:medium": 1,
"cid:phone:short": 1,
"cid:phone:vlong": 2,
"cid:pid:long": 1,
"cid:pid:medium": 1,
"cid:pid:short": 1,
"cid:pid:vlong": 1,
"device:cid:long": 200,
"device:cid:medium": 48,
"device:cid:short": 48,
"device:cid:vlong": 200,
"email:cid:long": 1,
"email:cid:medium": 1,
"email:cid:short": 1,
"email:cid:vlong": 1,
"email_cut:email:long": 1,
"email_cut:email:medium": 1,
"email_cut:email:short": 1,
"email_cut:email:vlong": 1,
"foreign_ip:cid:long": 0,
"foreign_ip:cid:medium": 0,
"foreign_ip:cid:short": 0,
"foreign_ip:cid:vlong": 0,
"parent_id:cid_selected:long": 0,
"parent_id:cid_selected:medium": 0,
"parent_id:cid_selected:short": 0,
"parent_id:cid_selected:vlong": 0,
"phone:cid:long": 1,
"phone:cid:medium": 1,
"phone:cid:short": 0,
"phone:cid:vlong": 1,
"pid:cid:long": 1,
"pid:cid:medium": 1,
"pid:cid:short": 1,
"pid:cid:vlong": 1,
"pid:ip:long": 1,
"pid:ip:medium": 1,
"pid:ip:short": 1,
"pid:ip:vlong": 1,
"ts:card:4r5cvbmo:long": 2,
"ts:card:4r5cvbmo:medium": 2,
"ts:card:4r5cvbmo:short": 0,
"ts:card:4r5cvbmo:vlong": 2,
"ts:cid:eczbwda6:long": 30,
"ts:cid:eczbwda6:medium": 3,
"ts:cid:eczbwda6:short": 1,
"ts:cid:eczbwda6:vlong": 103,
"ts:device:538153811775573131520373737116177585175837208784186:long": 32215,
"ts:device:538153811775573131520373737116177585175837208784186:medium": 1327,
"ts:device:538153811775573131520373737116177585175837208784186:short": 58,
"ts:device:538153811775573131520373737116177585175837208784186:vlong": 99894,
"ts:email:...:long": 30,
"ts:email:...:medium": 3,
"ts:email:...:short": 1,
"ts:email:...:vlong": 103,
"ts:email_cut:...": 30,
"ts:email_cut:...:medium": 3,
"ts:email_cut:...:short": 1,
```

```
      "ts:email_cut:...:vlong": 103,
      "ts:foreign_card:4r5cvbmo:long": 3,
      "ts:foreign_card:4r5cvbmo:medium": 3,
      "ts:foreign_card:4r5cvbmo:short": 1,
      "ts:foreign_card:4r5cvbmo:vlong": 3,
      "ts:ip:85.11.11.111:long": 6,
      "ts:ip:85.11.11.111:medium": 3,
      "ts:ip:85.11.11.111:short": 1,
      "ts:ip:85.11.11.111:vlong": 146,
      "ts:parent:card:long": 42525624,
      "ts:parent:card:medium": 802663,
      "ts:parent:card:short": 91700,
      "ts:parent:card:vlong": 115295888,
      "ts:phone:....:long": 22,
      "ts:phone:....:medium": 1,
      "ts:phone:....:short": 0,
      "ts:phone:...:vlong": 72,
      "ts:pid:4r5cvbmo:long": 3,
      "ts:pid:4r5cvbmo:medium": 3,
      "ts:pid:4r5cvbmo:short": 1,
      "ts:pid:4r5cvbmo:vlong": 3
    }
  }
}
```


Обновление события (updateEvent)

Обновление статуса ранее проверенного события.

Обратите внимание, что к этим запросам также могут применяться правила для того чтобы обновлять признаки. Например, если требуется учитывать только успешные операции или операции с подозрительными кодами ошибок.

PUT URL: `https://{customer}.cybertonica.ru:7499/api/v2.2/events/{channel}/{extid}?status={status}`

Замечание: при внедрении внутри периметра заказчика адрес может отличаться.

Запрос:

где:

`channel` - канал обновляемого события

`extid` - идентификатор события в канале

`status` - статус, одно из значений: `OK, FAILED, FRAUD`.

Тело запроса:

JSON

`t: long` - время в миллисекундах UTC, когда произошло обновление.

`code: int` - код результатам, например, ошибки.

`comment: string` - человеко-читаемое пояснение кода.

`is_authed: bool` (опционально) - флаг того, что была проведена попытка дополнительной авторизации события, например, 3DS/OTP.

Ответ: Успешный запрос завершается с кодом OK(200), тело пустое.

Возможные ошибки: 404 - обновление несуществующего события.

Получение результата скоринга (getEvent)

Получение результата скоринга для ранее проверенного события. Может быть полезно, в случае если на createEvent вернулся таймаут.

Метод идемпотентен (можно вызывать несколько раз).

Внимание! Поскольку тело запроса отсутствует в X-AF-Signature нужно подписать пустую строку.

GET URL: `https://{customer}.cybertonica.ru:7499/api/v2.2/events/{channel}/{extid}`

Запрос:

`channel: string` - канал события

`extid: string` - идентификатор события в канале

Ответ:

Аналогичен createEvent.

Возможные ошибки:

`401 Unauthorized`: Некорректная подпись. Проверьте алгоритм `X-AF-Team` и `X-AF-Signature`.

`404 Not found`: Событие не найдено.

`504 Gateway timeout`: Сервер Cybertonica в настоящее время недоступен

Проверка сетевой доступности

Проверяет сетевую доступность среды.

GET URL `https://{customer}.cybertonica.ru:7499/api/v2/ping`

Запрос: Пустой.

Ответ: Произвольная строка.

Возможные ошибки:

Ошибка установки соединения.

504 Gateway timeout

Описание полей

Токенизация

По соображениям информационной безопасности может быть нежелательна обработка некоторых полей в открытом виде, например, номер карты, номер паспорта, ФИО и т.д.

В этом случае, рекомендуется *до отправки* применять к этим полям криптографическую хеш функцию, например, SHA256.

В случае, если вы хотите иметь возможность сопоставлять данные с черными списками других заказчиков, то нужно применять `base64(SHA256(value))`.

Иначе, любую другую функцию, в том числе с криптографической солью.

Для некоторых полей можно применять специализированные схемы токенизации, например, для email - токенизировать только часть до `@`, для номера телефона - оставлять в открытом виде код страны и т.д.

Стандартная схема данных для платежа

Отправитель(src) и получатель(dst) идентифицируются парами `(parent, id)`, где: - `parent` - имя организации, где открыт счет, например, BIN. - `id` - идентификатор счета, например, токен карты.

В общем случае, платеж содержит как минимум следующие поля: - `t` - время платежа - `extid` - идентификатор платежа - `src_parent, src_id` - плательщик - `dst_parent, dst_id` - получатель - `amount` - сумма платежа - `currency` - валюта платежа

Внутри системы все платежи для удобства приводятся к единой базовой валюте.

Стандартизованные поля

`t: long` - время совершения события в миллисекундах UTC

`channel: string` - имя канала события

`extid: string` - идентификатор события в канале

`session_id: string` - идентификатор сессии от Fraudlock SDK (ранее это поле называлось `tid`).

`ip: string` - IP адрес клиентского устройства для определения GeoIP и т.д. Рекомендуется использовать `session_id` и Fraudlock SDK, но если такой возможности нет, то поле `ip` - это запасной вариант.

`src_parent: string` - идентификатор организации, где открыт счет отправителя.

`src_id: string` - идентификатор счета-отправителя.

`dst_parent: string` - идентификатор организации, где открыт счет получения.

`dst_id: string` - идентификатор счета-получателя.

`amount: int` - сумма платежа в минорных единицах (копейках/центах, например).

`exp: int` - экспонента валюты платежа, чаще всего 2 - копейки/центы.

`currency` - код валюты ISO 4217 (http://www.currency-iso.org/dam/downloads/lists/list_one.xls)

`mcc: int` - MCC-код мерчанта.

`email: string` - адрес электронной почты покупателя.

`phone: string` - телефон покупателя, только цифры.

Приложения

Приложение 1. Пример работы с API на Python

```
# Example how to use FraudSuite API.
# Note, that in real life you'd probably want to use async http.
# Tested: python 3.9
from typing import Optional, Dict, List
# you may prefer requests module, but we use standard module to limit 3rd party dependencies
import urllib.request
import json
import hmac
import hashlib
import time
import base64
import uuid


class FraudsuiteAPI(object):

    @staticmethod
    def sign(secret: str, msg: str) -> str:
        """ Computes X-AF-Signature accoring to the protocol """
        hashed_data = hmac.new(
            secret.encode(), msg.encode(), hashlib.sha1).digest()
        return base64.encodebytes(hashed_data).rstrip().decode("utf-8")

    def __init__(self,
                 protocol: str,   # usually 'https'
                 host: str,
                 port: int,       # usually 7499
                 team: str,
                 secret: str):
        """
        api = FraudsuiteAPI("https", "test.cybertonica.ru", 7499, "...your team...", "... your secret..
```

```
        """
        self.protocol = protocol
        self.host = host
        self.port = port
        self.team = team
        self.secret = secret

    def ping(self):
        """ Checks connectivity. """
        url = self._create_base_url() + 'api/v2/ping'
        req = urllib.request.Request(url)

        try:
            resp = urllib.request.urlopen(req)
            return resp.read()
        except urllib.error.HTTPError as e:
            raise Exception(
                f'ping() call failed, HTTP status {e.status}, reason: {e.read()}')

    def create_event(self, channel: str, sub_channel: str, payload: Dict) -> Dict:
        j = json.dumps(payload)
        signature = self.sign(self.secret, j)
        headers = {
            'Content-Type': 'application/json',
            'X-AF-Team': self.team,
            'X-AF-Signature': signature
        }
        url = self._create_base_url(
        ) + f'api/v2.2/events/{channel}?subChannel={sub_channel}'
        req = urllib.request.Request(
            url, headers=headers, data=j.encode('utf8'))
        try:
            resp = urllib.request.urlopen(req)
            return json.loads(resp.read())
        except urllib.error.HTTPError as e:
            raise Exception(
                f'create_event() call failed, HTTP status {e.status}, reason: {e.read()}')

    def update_event(self, channel: str, extid: str, status: str, payload: Dict) -> Dict:
        j = json.dumps(payload)
        signature = self.sign(self.secret, j)
        headers = {
            'Content-Type': 'application/json',
            'X-AF-Team': self.team,
            'X-AF-Signature': signature
        }
        url = self._create_base_url(
        ) + f'api/v2.2/events/{channel}/{extid}?status={status}'
        req = urllib.request.Request(
            url, headers=headers, data=j.encode('utf8'), method='PUT')
        try:
            resp = urllib.request.urlopen(req)
            # update_event does not return anything
            return None
        except urllib.error.HTTPError as e:
            raise Exception(
                f'update_event() call failed, HTTP status {e.status}, reason: {e.read()}')

    def get_event(self, channel: str, extid: str) -> Dict:
        # TODO: implement
        pass
```

```
    def _create_base_url(self) -> str:
        return f"{self.protocol}://{self.host}:{self.port}/"
```

Приложение 2. Чек-лист интеграции.

1) Сетевая доступность - проверить `/ping`. Измерить задержку (p99.9)
2) Отправляется createEvent.
    - Верная подпись.
    - Верный набор данных.
    - (опционально) успешно происходит сопоставление с сессией.
    - Обработка сценариев
        - ALLOW
        - CHALLENGE
        - DENY
    - обработка сценария ошибки
    - обработка сценария недоступности
3) Отправляется статус
4) Настроены ограничения на сетевом экране.